University of Heidelberg
Department of Psychology

January 2001

A Connectionist Approach to Human Planning

Diploma Thesis
of
Wolfram Schenck

Advisor and First Reviewer: Prof. Dr. Joachim Funke
Second Reviewer: Prof. Dr. Klaus Fiedler

Contact: mail@wolframschenck.de

Abstract

In the present thesis, a connectionist model (called EVA) was developed for the simulation of human planning behavior in the domain of Plan-A-Day. Plan-A-Day is a diagnostic instrument for the assessment of planning capabilities (Funke & Krüger, 1995), which provides a computer scenario in which the subjects must find an optimal sequence for several scheduled appointments by applying operators from a predefined set. EVA is designed to produce sequences of these operators which are applied within Plan-A-Day. In this way, EVA is intended to show the same kind of operationally defined planning behavior as human subjects do. EVA comprises three sub-networks of the backpropagation-class, two of which are hierarchical Elman networks. Based on the situated action account (Clark, 1997; Suchman, 1987), the input of EVA is for the most part a representation of the current state of the Plan-A-Day scenario. The data obtained by simulation runs is compared to empirical data obtained by a study with 45 human subjects. The comparison is carried out with regard to indicators reflecting planning performance, as well as characteristics of the process of planning. The fit between simulated and human data is good enough to allow certain theoretical claims. EVA provides evidence that simple pattern transformation devices like connectionist networks are capable of performing human-like planning, if these networks are thoroughly embedded into their environment, as it is demanded by the situated action account. EVA demonstrates that even conventional connectionist models are applicable to tasks from the field of high-level cognition like planning or problem-solving, which were mostly reserved for symbol-processing models until now.

Zusammenfassung

In der vorliegenden Arbeit wurde ein konnektionistisches Modell (genannt EVA) für die Simulation von menschlichem Planungsverhalten im Rahmen von Plan-A-Day entwickelt. Plan-A-Day ist ein diagnostisches Instrument für die Erfassung von Planungsfähigkeit (Funke & Krüger, 1995), in dem den Probanden ein Computerszenario vorgegeben wird. Innerhalb dieses Szenarios haben die Probanden die optimale Abfolge zahlreicher vorgegebener Termine herauszufinden, indem sie verschiedene Operatoren aus einer vordefinierten Menge verwenden. EVA wurde entwickelt, um Sequenzen solcher Operatoren zu erzeugen und diese innerhalb von Plan-A-Day anzuwenden. Dabei soll EVA ähnliches (durch Plan-A-Day operational definiertes) Planungsverhalten zeigen wie menschliche Probanden. EVA besteht aus drei konnektionistischen Netzwerken des Backpropagation-Typs; zwei davon sind hierarchische Elman-Netzwerke. Darüber hinaus basiert EVA auf dem Ansatz der “Situated action” (Clark, 1997; Suchman, 1987); so repräsentieren die Eingabeeinheiten von EVA zum größten Teil den aktuellen Zustand des Plan-A-Day-Szenarios. Die Daten, die in Simulationsläufen mit EVA gewonnen wurden, werden in der vorliegenden Arbeit mit empirischen Daten verglichen, die in einer Studie mit 45 menschlichen Probanden erhoben wurden. Der Vergleich bezieht sich auf verschiedene Indikatoren, die sowohl Planungsleistung als auch den Prozess der Planung erfassen. Die Übereinstimmung zwischen simulierten und empirischen Daten ist ausreichend, um bestimmte theoretische Aussagen zu unterstützen: EVA liefert Evidenz dafür, dass einfache Mustertransformation, wie sie von konnektionistischen Netzwerken vorgenommen wird, genügt, um menschliches Planungsverhalten zu modellieren, vorausgesetzt, dass diese Netzwerke sorgfältig in ihre Umwelt eingebettet werden, wie es der Ansatz der “Situated action” verlangt. Somit demonstriert EVA, dass sogar traditionelle konnektionistische Modelle auf Aufgaben aus dem Bereich der höheren kognitiven Funktionen wie Planen oder Problemlösen anwendbar sind, die bisher zum größten Teil symbolverarbeitenden Modellen vorbehalten waren.

Acknowledgments

It was a great help to me, that Stefanie Nellen (from the University of Heidelberg) gave me her kind allowance to use empirical data from a study she carried out. The whole process of model fitting is based on the human sample assessed in her study.

Bronwyn Jones (from the Humboldt University zu Berlin) had the patience to correct my imperfect English sentences. She took a lot of work upon herself, for which I am very grateful.

Furthermore, I appreciate the conversations Prof. Joachim Funke and I had, especially in the initial phase of the work, in which our discussions helped to clarify in which way connectionism and planning could come together.

Finally, I would like to thank both Prof. Joachim Funke and Prof. Klaus Fiedler for their interest and their readiness to give their expert opinions on this thesis.

Contents

> Artificial neural networks are fast but limited systems that, in effect, substitute pattern recognition for classical reasoning. As might be expected, this is both a boon and a burden. [...] A summary characterization might be “good at Frisbee, bad at logic” – a familiar profile indeed.
>
> (Clark, 1997, p. 60)

1 Introduction

Since the renewed arising of connectionism in the eighties, many different applications of connectionist models were presented. Among other disciplines, psychology is also involved in this development. During the last 20 years, the advocates of connectionism within psychology have presented many connectionist network models. These networks are aimed at the explanation or at least demonstration of cognitive functions, and partly these models have a high suggestive impact, as I experienced myself.

My first encounter with a connectionist model took place in 1989: It was a small network, consisting of only a few units, which was implemented on an Atari ST with black-and-white screen. This tiny network was able to remember surnames. Actually, this alone is not very exciting, but in those days, it was really impressive to observe the course of remembering. First, a fragment of one of the previously learned surnames was presented to the network. For example, “M_l__r”. Most likely, this task is very easy for you, dear reader, and you recognize the answer instantly: “Miller” – correct. However, look at the following word fragment “_pp_r__tl_”. Then, concentrate, until you know the solution. I guess, that at least after a few seconds you were able to identify the word. Something happened in your mind, and suddenly, the correct answer appeared in your consciousness. Before, maybe incomplete solutions were going around in your head (admittedly, this is the ideal course of this little introspective experiment). However, how did the billions of neurons in your head perform that task?

Let us return to the small network I was observing. After presentation of the prompt (“M_l__r”) to the network, its activation[1] began to circulate, and in every processing cycle, one could have a look at the current state of recollection. The following sequence developed: “M_l__r”, “M_l__r”, “M_l__r”, “Mil__r”, “Mil__r”, “Mill_r”, “Miller” – got it!

Thus, this small network showed a course of remembering similar to that used by at least some representatives of our species: The answer arose not suddenly, but in an iterative process, the answer became more and more clear. Even if this is not a proof at all, that the networks of biological neurons in our head complete word fragments in this way, this small artificial network model demonstrates at least, that neuron-like units are actually capable of performing tasks like this in a human-like style. Such models provide a primary understanding of how the real networks in our heads may work. This is the fascinating thing about connectionist models. By the way, the network I got to know in those days was most likely a kind of Hopfield network (Hopfield, 1982).

---

[1] In section 2, these technical terms will be clarified.

So far, connectionist models in psychology are mostly related to basic cognitive functions below the level of controlled and intentional information-processing. Phenomena from the fields of perception, attention, memory, or motor control have been intensively investigated within the connectionist framework. However, until today, cognitive functions that rely on controlled, intentional, and sequential thought processes have been rarely subject of connectionist modeling. Therefore, it is not very well understood, how connectionist networks could be applied to tasks from domains like problem solving or planning ("good at Frisbee, bad at logic", as Clark [1997, p. 60] writes). On the one side, even in the eighties one considered the application of connectionist networks to sequential thought processes (Rumelhart, Smolensky, et al., 1986), but on the other side, no concrete models were presented, and that is one of the reasons why the domain of high-level cognitive functions was taken up mostly by symbol-processing models like production systems (e.g., Anderson & Lebiere, 1998). In the discussion about the applicability of connectionist modeling at the end of the eighties, the opponents of connectionism saw its place at best in the domain of low-level cognitive functions (see Waloszek, 1996). Serious doubts arose regarding the general suitability of connectionist models for the domain of high-level cognitive functions (Barnden & Pollack, 1991).

Nevertheless, in this thesis, the challenge of developing a connectionist model of human planning behavior is accepted. Based on the mentioned considerations of Rumelhart, Smolensky, et al. (1986) on the one hand, and the related approaches of situated action (Suchman, 1987) and situated cognition (Clark, 1997) on the other hand, a standard model of connectionism is applied to the task of planning. The objective of this thesis is to demonstrate, that there is actually a place for connectionist models in the domain of controlled and intentional information-processing, even for traditional models, which originated in the eighties.

The planning task for which a connectionist model is developed in this thesis is taken from a diagnostic instrument, called "Plan-A-Day" (Funke & Krüger, 1995). In "Plan-A-Day", the subjects have to work out a plan in a businesslike scenario. Several appointments are scheduled for a certain day, and the subjects must arrange these appointments in a way so that they are able to fulfill as many tasks as possible. In doing so, they have to consider many properties of the artificial "Plan-A-Day"-world, for example movement times between different locations with scheduled appointments. In section 3 of this thesis, "Plan-A-Day" is described in more detail, while section 2 provides both an introduction into connectionism and into the field of planning.

Then, in section 4, the method of creating a connectionist model capable of planning within the "Plan-A-Day"-world is described. This section covers many topics, beginning with the theoretical foundations underlying the present model, and ending up with a close view onto the course of network training. Section 5 is dedicated to the subject of model fitting. Furthermore, a study for gaining empirical data from human subjects is described. In section 6, the results are presented. Mainly, the planning behavior of the connectionist model is compared to the planning behavior of human subjects from the empirical study. Moreover, this section deals with a first attempt at validation. Section 7, finally, comprises the discussion of the thesis' approach in light of the underlying theoretical assumptions and the actual results.

2 Connectionism and Planning

2.1 Connectionism

2.1.1 Paradigm of Connectionism

The notion of "connectionism" stands for a broad range of different models which have one thing in common: They offer an approach to information processing that is inspired by the architecture of the brain. Especially, these models cover the aspect of parallel distributed processing in that they provide quite a large number of simple nodes which are highly interconnected and only capable of very simple input-output-transformations. These nodes are often called "neurons" in analogy to biological neurons, but actually they model real neurons on a quite abstract level. Therefore research in connectionism deals with the simulation of artificial neural networks and the examination of their properties. The goal of such neural network research can differ widely depending on the discipline within one works.

Computer science, mathematics, physics, electrical and computer engineering, biology, medicine, philosophy, and last but not least, psychology, can claim their interest in connectionism. Working on neural networks is a quite interdisciplinary affair, and often results from one discipline are useful for the whole field. In the history of neural networks[2], this broad approach has led to a parallel development in the related disciplines with a changing focus of attention throughout the years. One important motivation for connectionist modeling lies in their capability of brain-style computation (Rumelhart, 1989). On the one hand, this offers the possibility of benefiting from natural examples in the development of technical solutions, as it is sometimes the case in engineering; e.g., computer scientists can use neural networks for replacing conventional algorithms. On the other hand, in cognitive science, through neural networks one can use a class of models which are inspired by the architecture of the brain itself for explaining human information processing capabilities.

2.1.1.1 Basic Architecture of Neural Networks

In spite of the fact that connectionist models cover a broad range of applications and different purposes, their basic architecture is quite similar. Therefore, the following generic description covers a large part of the field although many details can differ for a specific model.

Every network consists of a certain number of neurons[3] which are connected to each other (see fig. 2.1). The topology of the network determines which connections exist. For example, there could be a group of neurons which are fully interconnected to another group of neurons, but not within each group. Every neuron has several input lines by which it receives input

[2] The term "connectionism" was first introduced in 1981 (Feldman, 1981; cited in Zell, 1997).

[3] The term "certain number" is not intended to imply that this number cannot be subject to change while the network is working. E.g. the "cascade correlation architecture" (Fahlman & Lebiere, 1990) is based on the idea of adding new neurons to the network in dependence on the learning success.

values from other neurons. In one processing step each neuron transforms its input values to one output value that is fed forward to other neurons for the next processing step. This input-output-transformation is often carried out by a formula like this:

$$a_j = f_{act}\left(\sum_{i=1}^{n} w_{ij} o_i - \theta_j\right) \tag{2.1}$$

a_j is the activation strength of neuron j; o_i is the output value of neuron i; w_{ij} is the connection weight for input line i; θ_j is the threshold of neuron j; f_{act} is the activation function that transforms the weighed and added input into the activation value. For simplification I assume that the output o_j of neuron j is equal to its activation a_j.

For the activation function f_{act} two different types are very common: One possibility is to use a threshold value function that produces either 0 or 1, depending on whether the argument is larger than zero. Alternatively, one can employ the logistic function $f = \frac{1}{1+e^{-x}}$ with a value range between 0 and 1. If prefered, one can modify these functions in a way that changes their value range, e.g. to {-1, +1} or [-1; +1]. Furthermore, in some models a linear activation function or even the identity function is used.

A neuron is characterized by its activation function, its threshold and its input weights. Its activation is subject to modification in every processing step through changes in the input

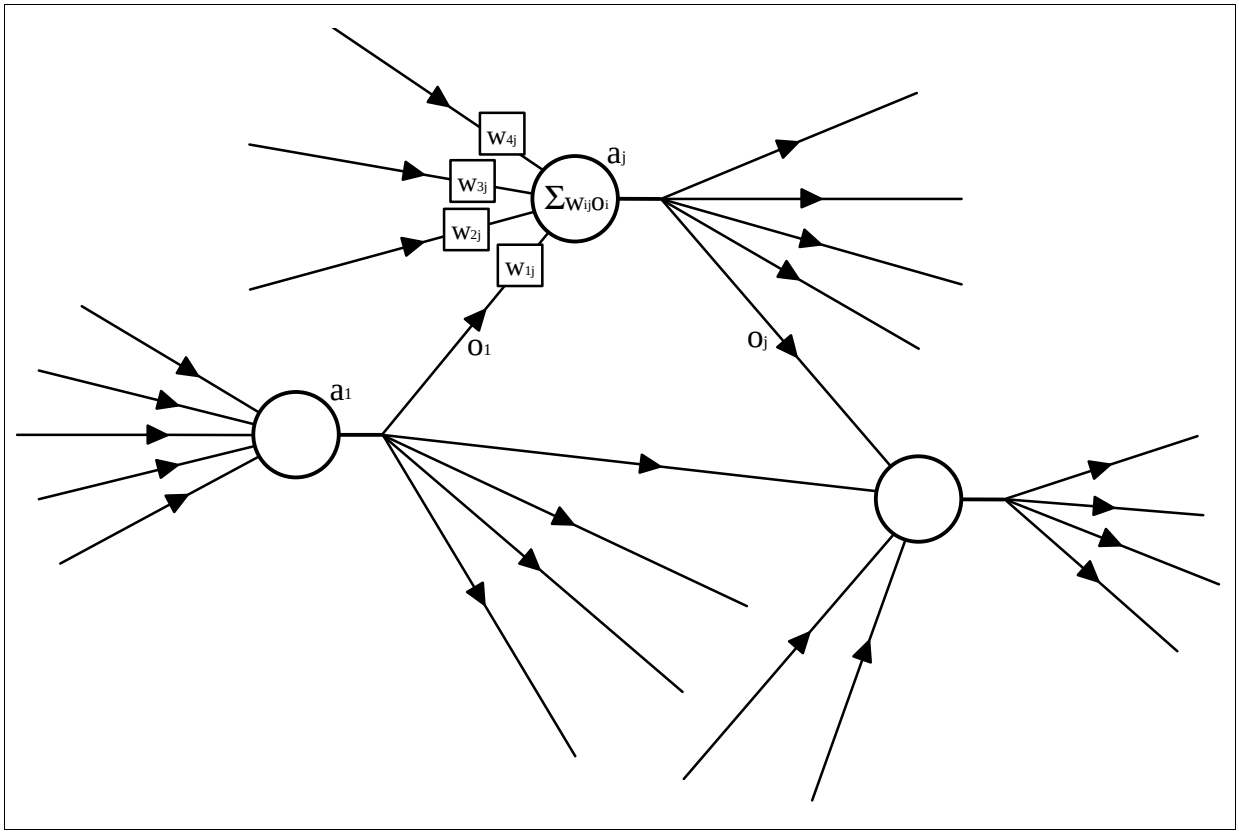

Fig. 2.1: Part of a connectionist network. Three units are shown as an example. Arrows coming from nowhere (or going to nowhere) are supposed to be connections from (to) other units not shown in the figure. Unit j gets input from four other units; each input line has an input weight w_{ij}. The activation a_j of the unit is determined through a transformation of the sum of weighted inputs (see text). Afterwards, the activation a_j is propagated forward to other units as output o_j.

values. Therefore, the whole network can be thought of as a complex interdependent dynamic system the elements of which (the neurons) change their state (activation) according to the changing states of those elements from which they receive information. There is no global control over the system; every neuron only receives a small local piece of information about the state of the network and uses only this part for updating its own activation. The "behavior" of the network is an emergent phenomenon, which arises from the simultaneous work of its neurons – a kind of self-organization takes place.

The whole set of input weights of the network can be written in the form of a matrix which is often called *W*. Each element w_{ij} is a connection weight from neuron i to neuron j.[4] If there is no connection from neuron i to neuron j, then w_{ij} equals zero.[5] The weights determine the behavior of a single neuron as well as that of the whole network and serve as its memory. Connectionist models differ from conventional information processing models especially in one respect: There is no subdivision into several units which carry out only part of the work (memory storage, regulation of the information flow, execution of operators, etc.). The memory of a connectionist system is distributed over all connection weights. When the system is working, the changing pattern of neural activation reflects the ongoing information processing, which is directly determined by the weights' values.

2.1.1.2 Localist vs. Distributed Representations

There are many possibilities to categorize connectionist models. One important distinction refers to the kind of representation used for the processing units (the neurons). In a localist representation, each processing unit stands for an entire concept or another large meaningful entity. In this respect, a localist representation resembles a traditional symbolic representation. One can think of such units as hypothesis detectors. Each unit's activation strength can be taken as an indicator of the strength of the concept being represented (Elman et al., 1996).

In contrast, in a distributed representation, all concepts are represented by a common set of units. Which concept is active depends on the current pattern of activation across the entire ensemble. Every unit represents a kind of microfeature or even less than that; in some models it is nearly impossible to assign any distinct meaning to certain units. It is assumed that distributed representations have more similarity to natural neural networks in the brain than localist representations have.

Often one finds a mixture of both representations. In so-called "backpropagation networks" the model consists of several layers of neurons which are arranged one behind the other. The first layer receives input from the outside world and the last layer produces the appropriate output. The hidden layers between the first and last layer serve as pattern transformators. The

[4] In some models, the threshold θ_j is replaced by an "on-neuron" and an input weight corresponding to this neuron. Then even the threshold θ_j is part of the matrix *W*.

[5] As a biological analogy, the connection weights can be compared with the synapses' strengths. As the strength of a synapse determines how strong the pre-synaptical neuron can influence the post-synaptical neuron, the connection weight determines how much the output of neuron i contributes to the total activation of neuron j. A connection weight of value zero means that no synapse exists at all.

experimenter can assign certain meanings and concepts to the neurons of the input layer and the output layer (creating a localist representation), but especially in large networks, the function and meaning of hidden neurons cannot be reduced to any clear-cut area. Instead, one finds a completely distributed representation. The question of how such representations can develop leads to the next section about learning in neural networks.

2.1.1.3 Learning in Neural Networks

Connectionist models are used for a wide range of applications. Usually, a network's task is to transform a vector of activation values (the "input pattern") to another vector of activation values (the "output pattern"). In some models the input pattern is assigned to the same neurons which are later used to read out the output pattern, in other models input and output patterns are related to different sets of neurons. Belonging to the first class of models are, for example, the Hopfield network (Hopfield, 1982) and the Boltzmann machine (Hinton & Sejnowski, 1986). The second class includes the bi-directional associative memory (BAM) (Kosko, 1987) and the backpropagation network (Rumelhart, Hinton, & Williams, 1986). In this section the question of how a network can acquire a meaningful input-output-assignment is addressed.

In the course of research on connectionist models several learning rules were developed for this purpose. The common goal is to modify the connection weights w_{ij} in a way that allows the network to perform its task as well as possible. As an example, three important learning principles are presented in the following paragraphs: Hebbian learning, the Widrow-Hoff rule, and the generalized delta rule. Other well-known learning procedures (e.g., for self organizing maps [Kohonen, 1982], for Hopfield networks [Hopfield, 1982], or for bi-directional associative memory [Kosko, 1987]) are omitted because they are not directly related to the model which is discussed in this thesis.

Hebbian learning

The idea of Hebbian learning was first formulated for biological networks by Donald O. Hebb: "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased." (Hebb, 1949, p. 62; cited in Elman et al., 1996)

This rule is easy to apply to artificial neural networks. Correlated activity of two connected neurons should lead to an increasing connection weight between them. In mathematical form:

$$\Delta w_{ij} = \eta o_i a_j \tag{2.2}$$

Δw_{ij} is the change in the weight of the connection from sending neuron i to receiving neuron j, η is a small constant of proportionality (the learning rate), o_i is the output of neuron i, a_j is the activation of neuron j.

The Hebb rule only uses local information. There is no need for any global evaluation of network performance. Also, no teacher is required to tell the network in which direction it should modify its weights (unsupervised learning). Due to these two aspects, Hebbian learning seems to have biological plausibility.

Hebbian learning has been studied extensively and is widely used in modeling today. However, a severe limitation of this approach exists. With the Hebb rule only pair-wise correlations can be learned – higher-order correlations are beyond its scope.

The Widrow-Hoff rule

The Widrow-Hoff rule (Widrow & Hoff, 1960; cited in Waloszek, 1996) offers a solution to the dilemma of Hebbian learning. Sacrificing biological plausibility, an external teacher is provided for the network who keeps track of the desired, as well as of the produced output pattern (supervised learning). The difference between the actual activation and the desired activation determines the modification of connection weights:

$$\Delta w_{ij} = \eta o_i (t_j - a_j) \tag{2.3}$$

Δw_{ij} is the change in the weight of the connection from sending neuron i to receiving neuron j, η is the learning rate, o_i is the output of neuron i, a_j is the actual activation of neuron j, t_j is the desired activation of neuron j.

In general, the Widrow-Hoff rule (often called delta rule) is applied to networks with two layers of neurons and linear activation functions. Within each layer there are no connections between neurons, but every neuron in the second layer receives input from all neurons of the first layer. The task of such a network is to associate the input pattern that is presented to the first layer with the appropriate activation pattern in the second layer (the output pattern).

Unfortunately, there are some theoretical limitations to this class of networks. If one wishes such a network to learn several input-output assignments simultaneously (which is the regular case), one has to consider some serious restrictions on the choice of different pattern pairs. Elman et al. (1996, p. 59) write: "Two-layer networks are rather like S-R pairs in classical psychology. What is required is something between input and output that allows for internal (and abstract) representations."

The generalized delta rule

What is needed to overcome the weaknesses of a two-layer network are so-called "hidden layers". The architecture of a traditional multi-layer (feed-forward) network provides one input layer, one hidden layer, and one output layer (see fig. 2.2). Every neuron propagates its output forward to every neuron of the following layer, but within each layer there are no connections between neurons at all. The neurons of the hidden layer constitute an internal representation of the input in relation to the desired output. With such a network nearly every set of input-output associations can be realized simultaneously. In other words: A single hidden layer gives

networks the power to solve essentially every problem[6], as Hornik et al. (1989) showed by a mathematical proof. Unfortunately, this proof is not constructive. It only states that for every problem a three-layer network with the appropriate weights does exist. However, it does not say anything about the number of neurons needed in the hidden layer, nor about the correct values for the weights.

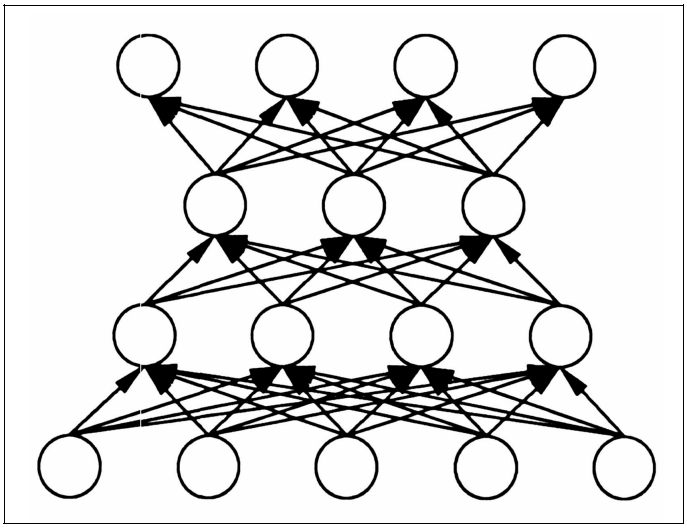

Fig. 2.2: A multi-layer feed-forward network with two hidden layers. In such an architecture, every unit of each layer gets input from every unit of the preceding layer. Within layers, no connections exist (fig. taken from Zell, 1997, p. 73).

At least for the latter problem – finding the correct weight values – a solution exists: The generalized delta rule. This rule does not guarantee that one finds the optimal set of weights for a certain problem, but it offers a more or less good approximation.

The generalized delta rule was first published by Werbos (1974; cited in Zell, 1997), but it did not become well-known until the work of Rumelhart, Hinton, and Williams (1986) was published. The idea behind this approach is to generalize the conventional delta rule so that it can be used for multi-layer networks. Therefore, the basic formula is quite similar to the delta rule:

$$\Delta w_{ij} = \eta o_i \delta_j \tag{2.4}$$

Δw_{ij} is the change in the weight of the connection from sending neuron i to receiving neuron j, η is the learning rate, o_i is the output of neuron i, δ_j is the error signal of neuron j.

The error signal δ_j of neuron j is calculated in the following way:

$$\delta_j = \begin{cases} f'_{act}(\sum_i o_i w_{ij})(t_j - o_j) \text{ if j is an output neuron} \\ f'_{act}(\sum_i o_i w_{ij}) \sum_k (\delta_k w_{jk}) \text{ if j is a hidden neuron} \end{cases} \tag{2.5}$$

w_{ij} is a weight of the connection from sending neuron i to receiving neuron j, w_{jk} is a weight of the connection from sending neuron j to receiving neuron k, o_i is the output of neuron i (located in the preceding layer relative to neuron j), o_j is the output of neuron j, t_j is the desired output of neuron j, δ_k is the error signal of neuron k (located in the following layer relative to neuron j), f_{act}' is the first derivation of the activation function f_{act}.

[6] More precisely: Multi-layer networks are capable of approximating any Borel measurable function from one finite dimensional finite space to another, as Hornik et al. (1989) showed.

If one uses the logistic function as an activation function (see section 2.1.1.1), one can calculate the first derivation of f_{act} by

$$f'_{Log.Act.}(x) = f_{Log.Act.}(x) \cdot (1 - f_{Log.Act.}(x)). \quad (2.6)$$

With the given formulas, one can train any multi-layer network (even with several hidden layers) to reproduce certain input-output assignments. This training procedure is called "back-propagation", because the output error ($t_j - o_j$) is propagated back into the network by the error signals δ_j. The weight changes Δw_{ij} can be carried out after the presentation of each input-output-pattern-pair (online training), or they can be added up until the whole set of pattern pairs has been presented to the network (batch training). Depending on the complexity of the problem many training epochs[7] may be necessary.

Backpropagation belongs to the family of gradient descent procedures. The basic idea behind the backpropagation procedure is to minimize the global error function E (which comprises the error over all output neurons and output patterns). For that purpose, the weight changes Δw_{ij} move in the opposite direction to the partial gradient $\frac{\partial}{\partial w_{ij}} E(W)$ (W is the matrix of all network weights). If one prefers a more lively picture: The function $E(W)$ is like a landscape in the weight space. The training procedure attempts to find the deepest point in this landscape (like a rolling ball). Unfortunately, the "ball" can end up in a local minimum and thus, not ever be able to find the global minimum. This is a severe problem for backpropagation, and many modifications of the standard backpropagation procedure attempt to solve it.

Another problem is the ability of trained networks to generalize to new input-output-pattern-pairs they have never seen before, but which belong to the same class of problems the network has learned through the training patterns. In this respect, it is helpful to think of every network as a specification of an inductive hypothesis about the assignment of input to output patterns. For most problems, there are enough degrees of freedom to allow different solutions – some with good generalization capabilities and some without. Rumelhart (1989) suggests some ways of finding a good network in this respect.

The biological plausibility of backpropagation is called into question by the "external teacher" who is necessary for the learning procedure. On the other hand, the exclusive use of local information for learning, as well as for pattern transformation is reasonable in analogy to biological systems.

2.1.2 Connectionism in Psychology

Connectionist models have been applied to many different research topics within psychology during the last twenty years. These models mostly yield insight into the implementational level. They show how certain cognitive functions could be realized by neural networks. However, the implementational level can only seldom be separated from the functional level, so that

[7] The complete presentation of all pattern pairs together with the corresponding weight adjustments is called "epoch".

connectionist models are often accompanied by functional claims.[8] Sometimes these models even make new predictions about human cognitive performance which go beyond predictions made by previous traditional psychological models. The following overview of connectionist literature is not intended to be a complete presentation of the field. It only provides some examples of psychological applications of connectionist models to show the broad variety of investigated phenomena.

Hinton and Shallice developed a model of deep dyslexia and semantic-accesss dyslexia that was based on an iterative variant of backpropagation (Hinton & Shallice, 1991). Because of the recurrent and iterative characteristics of their network, its dynamic behavior exhibited several attractors – stable states to which the network was converging. The authors showed that so-called "graceful degradation" of their network produced phenomena similar to the phenomena shown by human subjects with deep dyslexia. Graceful degradation means that several connections or units within the network are disturbed or removed, a technique often used to demonstrate the stability of distributed representations in connectionist networks. In addition, this technique can be employed in the field of psychic disorders as Hinton and Shallice did. Another example is the study of Hoffman et al. (1995) in which a recurrent multi-layer network was subject to graceful degradation. The simulation showed certain speech perception impairments and illusions such as those experienced by schizophrenic patients. The authors concluded from the characteristics of their simulated network that certain changes must have taken place in the neural systems underlying verbal working memory of schizophrenic patients. This study is an example of a connectionist model from which strong hypotheses were derived on the level of real neural systems.

In his original work on a special kind of recurrent multi-layer network, Elman (1990) applied his network model to various tasks, among other things to the identification of word boundaries in the stream of language. He pointed out that such a simple recurrent network is able to extract the necessary information from the input signal. In a similar way, he demonstrated how lexical classes were derived from word order. Tasks and network applications like these are especially interesting for developmental psychologists who are interested in the course of learning. In such models, it is shown how learning of simple natural-like stimuli can lead to a meaningful internal representation within the network. In his study, Elman presented simple sentences to a network with the task of predicting the next word in the sentence. As a result of the learning process, the network developed an internal representation for the hidden unit activation vectors, which reflected the hierarchical order of the lexical classes to which the occurring words belonged.

An alternative approach is taken in the work of Cohen et al. (1992), which addresses the question of automaticity and attention. In their study, two network models are presented, each of which are composed only of a small set of units. The connections in these networks were not a result of a training process but instead analytically determined. The authors demonstrate how certain issues concerning automaticity can be understood in terms of parallel distributed

[8] For the discussion about the "question of levels" see Broadbent's comment on a connectionist memory model by McClelland and Rumelhart (Broadbent, 1985; McClelland & Rumelhart, 1985).

processing. Studies like this are valuable for psychology because they show, in principle, how such cognitive functions can be implemented by neural networks, even if nobody would claim that exactly the proposed network is working in the human brain.

Blankenberger (1992) outlines several simulations of relatively small multi-layer networks (about twenty units) some of which are recurrent. With these networks, he was able to reproduce certain phenomena in the field of mental comparison processes. The distribution of reaction times of human subjects in number comparison tasks was compared to the distribution of iterative cycles which trained networks needed to reach a solution for the same tasks. With a certain kind of input representation and a certain kind of network architecture, Blankenberger achieved a satisfying fit between simulation and experimental data. This study is a good example for connectionist modeling of a special cognitive function.

Another model in the field of multi-layer networks is ALCOVE (Kruschke, 1992; Kruschke, 1993). Kruschke developed ALCOVE as an improvement of standard backpropagation networks, which show several shortcomings in the field of category learning when compared to human beings. ALCOVE is a three-layer feed-forward network in which the hidden units are replaced by nodes of a type of radial basis function (RBF). When one compares the category learning course between the simulation and human subjects, ALCOVE showed a much better fit than standard backpropagation networks. ALCOVE demonstrates how psychological models can influence the development and construction of network architectures in advance; in the case of ALCOVE the underlying model is Nosofsky's generalized context model (Nosofsky, 1986).

Leaving the field of multi-layer networks, one can find much simpler models which, nevertheless, do not lack explanatory power. One example is the study of McClelland and Rumelhart which is concerned with distributed memory (McClelland & Rumelhart, 1985). A small network of interconnected units is used to demonstrate how prototypes can be learned from exemplars (only by the delta rule). This study belongs to a large field within connectionism related to memory, within which many different models were developed.

Apart from general cognitive psychology, connectionist modeling is also relevant for social cognition, as different contributions have shown (Read et al., 1997; Smith, 1996). The authors mainly refer to constraint satisfaction processes. Constraint satisfaction represents a special view of the processes running in connectionist networks. In light of this view, individual aspects of impression formation and causal attribution, cognitive consistency, and goal-directed behavior become the subjects of connectionist modeling. Another successfully simulated phenomenon relevant to social cognition is priming in various forms.

What has not been mentioned so far are high-level cognitive functions like those which are investigated in psychology (e.g. reasoning, problem solving, or planning). Nearly every connectionist model in psychology deals either with low-level cognitive functions or with a small aspect of a more complex function. This can be seen in the aforementioned studies which are related to phenomena like semantic access, speech impairments, extracting temporal information from input streams, automaticity, mental comparisons, category learning, prototype learning, and various individual aspects of social cognition. In general, the term low-level cognition refers, in this thesis, to cognitive functions below the level of intentional and controlled

information processing, the term high-level cognition will be used for functions above this level. In contrast to those low-level functions, up until now intentional and controlled information processing were mainly modeled and simulated by symbolic systems – systems that operate with languagelike symbols (concepts) that can be combined in structured ways to encode propositions or to form complex representations.[9] At the theoretical level, the opponents of connectionism claim that this symbolic approach is the only appropriate way for explaining high-level cognition (e.g., Fodor & Pylyshyn, 1988). On the other hand, the advocates of connectionism view connectionist modeling as suitable for all types of human cognition (e.g., Smolensky, 1988), even if they appreciate the usefulness of symbolic models in cognitive science. For a survey of this discussion see Waloszek (1996). Unfortunately, standard connectionist models encounter many problems with regard to high-level cognitive functions as is shown in section 2.3. Based on this background, this thesis attempts to develop a connectionist model of planning, which is quite a challenging affair.

2.2 Planning

2.2.1 Planning in Psychology

The place of planning in psychology is close to the field of action regulation. In many models of action regulation a kind of planning is involved to explain the succession of action steps. Since action is goal-directed, in contrast to behavior, a plan consists of several action steps necessary for reaching a certain goal. Moreover, in a plan the order of steps is prescribed. Models of action regulation differ in the degree to which the phases of plan preparation and plan execution are separated and to which situational circumstances are favored above planning processes in explaining action. The most extreme position is taken by the behavioristic school of thought, which rejects the notion of planning completely. Models from cognitive psychology weigh the contribution of planning and situation toward action in different ways: On the one hand, the notion of "situated action" (e.g., Suchman, 1987) stresses situational influences. On the other hand, models like the one by Dörner (1989) emphasize the cognitive effort which precedes and accompanies goal-directed action.

The differentiation between plan preparation and plan execution is elaborated in the work of Funke and Glodowski (1990). They propose several basic competencies for both stages. The differentiation between these stages becomes clear in the definition of planning provided by Funke and Fritz (1995): "Planning means: Intellectual sketch of a goal-directed action sequence, which may take place on different levels of resolution under consideration of spatial, chronological, material, and logical constraints, given the current level of skills and knowledge. Moreover, planning means monitoring the prepared plan during its execution with the possibility of revising it or of terminating it. Working out the plan and executing it may overlap in

[9] An example for the symbolic approach to high-level cognitive functions is the production system ACT-R (Anderson & Lebiere, 1998).

time. The planning process is finished when the (revised) goal is reached or abandoned." (Funke & Fritz, 1995, p. 29; translation by the author)

This definition shows that the place of planning is somewhere between action (plan execution) and problem solving (plan preparation as intellectual sketch). While the former aspect is quite obvious in light of the remarks so far, the latter aspect needs some elaboration. The notion of problem solving is used for mental processes which are engaged in transforming an undesirable initial state into a desired goal state. These states may be mental states (e.g., when thinking about a riddle which is completely represented and solved within the mind) or real world states (e.g., when thinking about building a new company headquarters). In the conception of Newell and Simon (1972; cited in Opwis, 1996) all possible states of a problem form its problem space (see also Anderson, 1996). By use of operators, the problem solver is able to move from problem state to problem state, ideally reaching the goal state at the end. Thus, there exists a strong analogy between plan preparation and problem solving. In plan preparation one wants to find a sequence of actions which leads to the goal state, in problem solving one wants to find a sequence of operators which leads to the goal state as well. Before I go deeper into this analogy, one first distinction must be mentioned. The notion of problem solving is not applied to problems which can be solved just by one step of memory retrieval (Medin & Ross, 1992). However, simple plans can be sketched in this way. Therefore, one can remark that planning without problem solving does exist, although often plan preparation is an activity quite analogous to that of problem solving.

Vice versa, not all problem solving is plan preparation, because the latter is restricted to one type of operator: Actions. In contrast, e.g. proving a mathematical theorem needs operators, which are quite different from actions. Thus, if one distinguishes between the stages of plan preparation and plan execution, the former stage can be identified to a great extent with the subset of problem solving concerning actions. As in problem solving, planning tasks can also be plagued by "planning problems", which are ill-defined or well-defined, as well as by the differentiation between knowledge-lean and knowledge-rich problems (Opwis, 1996).

The aforementioned definition of planning points out, that "working out the plan and executing it may overlap in time" (Funke & Fritz, 1995, p. 29). This complicates the subject, because the sketch of the plan may be revised due to new information about the state of the environment or other sources. At first sight, it seems that plan preparation which is intermixed with execution cannot be compared with problem solving. However, if the notion of "situated cognition" (Clark, 1997) is included in these considerations, the physical environment becomes important for problem solving too. For many problems, the environment is used as an external aid to find a solution (e.g., paper and pencil may be used to take a note of intermediate steps or as an external memory extension). Operators may modify the environment and change the part of the problem state which is manifested in the real world. This change may lead to new possibilities for developing further ideas and operators. Clark (1997) designates this exploitation of external structure in the process of problem solving as "scaffolding". Thus, even the stage of plan execution, at least when it incorporates plan revision, has some similarity to general problem solving. Still, one difference remains: Actions which were carried out cannot be undone

afterwards, while, theoretically, in problems which do not involve actions every known state in problem space is always attainable.

Even if it is useful for some models to separate the stages of plan preparation and plan execution, many real world planning problems require that these phases are strongly interconnected with each other because of missing information and uncertainty of the environment. As Hertzberg (1989) points out, many plans cannot be sketched entirely. There have to be "open slots" and sensory actions to obtain current information in specific situations when plan execution is in progress. Independent of this specific information, the plan must be specified more exactly or revised. This topic is taken into consideration by many psychological models which provide feedback and control loops like the TOTE-unit (Dörner, 1989; Werbik, 1978; cited in Funke & Fritz, 1995).

After all, the comparison between problem solving and planning shows that planning corresponds to a special subset of problem solving which calls for one type of operators, namely actions. Beyond it, as pointed out before, planning has some special properties in which it differs from problem solving in general. This analogy holds but for the case of simple planning tasks which only need memory retrieval of prepared solutions. These already existing action sequences are called "scripts" by Schank and Abelson (1977). According to Schank and Abelson, a need for "real planning" exists only in novel situations.

So far I have considered the mental side of planning and its relation to problem solving. Moreover, as mentioned in the beginning, planning is closely related to action regulation and is used as an explanation of why human beings show certain actions in a certain sequence. An alternative or additional explanation lies in the notion of "situated action" (Greeno & Moore, 1993; Norman, 1993; Suchman, 1987; Vera & Simon, 1993). This approach emphasizes how strongly actions are triggered by situational circumstances and cues.[10] The gap between situation and action is filled by rules which the agent has learned by prior experience (Hertzberg, 1996). In close relationship to the notion of "situated cognition", it is proposed that planning heavily depends on the stage of execution, if planning takes place at all. During execution the general plan, which looks like a loose framework, is filled with concrete actions in dependence on the situation. Suchman (1987, p. 188) writes: "While plans *can be* elaborated indefinitely, they elaborate actions just to the level that elaboration is useful; they are vague with respect to the details of action precisely at the level at which it makes sense to forego abstract representation, and rely on the availability of a particular, embodied response."

2.2.2 Planning in Artificial Intelligence

The definition of planning in Artificial Intelligence (AI) is more technical than in psychology, as stated by Hertzberg (1996, p. 501; translation by the author): "In AI a plan is a structure which contains representations of actions and goals; its purpose is to reason about the effect of

[10] In some respect, a similar approach was even taken by Lewin (1935) who introduced the notion of "valence" into psychology. According to his field theory, the valence of situational circumstances contributes to the motivational force which drives the person to execute certain actions.

future actions and to influence the goal-directed acting of an actor. Planning designates the actual production of a plan and, at the same time, a subsection of AI."

Inder (1996, p. 23) gives a shorter definition and distinguishes planning from problem solving at the same time: "In the AI literature, planning refers to determining a sequence of actions you know how to perform that will achieve a particular objective. Problem solving is finding a plan for a task in an abstract domain."

As in psychology, planning involves finding a sequence of actions that will transform an initial state into a goal state. The difference is that the AI concept of planning only involves the stage of plan production or preparation as I called it in section 2.2.1. The execution of the plan is not mentioned in the definitions presented so far. This corresponds to the notion of "classical planning" as it was very popular for many years in AI. Before I go deeper into this subject, one remark about the relation of planning and problem solving must be made. Roughly spoken, in psychology, planning may be understood as a subset of problem solving (see section 2.2.1). Inder (1996) shows a reverse understanding (for the domain of AI): In his definition problem solving is a special class of planning. This demonstrates how difficult it is to draw a clear line between problem solving and planning. Thus, how one understands these notions depends heavily on the definitions and focal points one has chosen.

Classical planning

As noted, in AI planning refers mainly to the stage of plan production or preparation. This is one of the constraints which developed from the early work on planning. To reduce the complexity of planning tasks so that artificial systems were able to produce plans for them, several restricting preconditions crystallized within the domain. These constraints define the frame of "classical planning" which is now well understood. In the following the ten most relevant of the thirteen preconditions stated by Hertzberg (1996) are presented:

(i) There exists exactly one planning actor.
(ii) It is possible to represent the relevant part of the world in states. These states are complete snapshots of the world.
(iii) State transformations by planned actions are the only form in which time is represented within the range of application.
(iv) Planning and plan execution are carried out one after another.
(v) Complete information about the facts within the "world" are available during planning as well as during the completion of plan execution.
(vi) The effects of an action are deterministic and context-free. That means, they are identical for every state in which the action is executable.
(vii) During plan execution the world is only changed by the actions of the actor who is guided by the plan.
(viii) The objectives for the resulting plan are explicitly stated; they are consistent and can be achieved by the known actions.

(ix) A ready-made plan has to achieve all its objectives; every action it consists of has to be applicable.

(x) The computing time for the production of a plan is not relevant for the assessment of its quality.

These preconditions demonstrate a static view of the world in which the actor is operating. Nothing can interfere with his actions. Beyond it, the range of the world is limited so that it fits into exactly defined states. One common-known example for such a "world" is the blocks world (e.g., in Hertzberg, 1989). The standard version consists of a table on which several cubes are placed. There is a robotic grip hand that is able to pick up one cube at a time. Any cube is either placed directly on the table surface or on one other cube or is held by the grip hand. The grip hand is capable of several pick-up- and put-down-operations. The planning problem consists of finding a sequence of actions for the grip hand so that an initial arrangement of cubes can be changed into the desired goal arrangement.

The blocks world was used in many AI planning systems to develop and demonstrate appropriate planning techniques in such a restricted domain. One of the first planning systems was STRIPS (Stanford Research Institute Problem Solver) (Fikes & Nilsson, 1971; cited in Inder, 1996), which generated plans for a robot, Shakey, that could move around between a number of rooms, pushing boxes around and carrying out a small number of other actions. Such a planning system provides a formal structure with which states can be described (e.g., the initial and the goal state of the world) and applicable operators/actions[11]. Beyond it, procedures must be available in order to produce a plan from the information about states and operators which the user has defined. The special feature of STRIPS is that states are described using a set of formulas in first-order predicate calculus. In brief, this formalism gave STRIPS the possibility of describing states mainly in respect to their difference to former states. This allows much shorter state descriptions. Besides, operators can be expressed as the changes they make to the set of formulas describing the state in which they are applied. As method for determining the optimal plan, STRIPS basically used the means-ends analysis. In means-ends analysis one looks first at the features of the goal state. Then one searches for operators with which one can realize the features not present in the current state. When one has found one or several operators, the preconditions these operators require for execution become sub-goals for the next cycle of analysis. The analysis ends when one has found a complete path from the goal state back to the initial state.

Beside the strong points of STRIPS, there are also several weaknesses which have led to many further developments and alternative planning systems during the decades following STRIPS. The focal point in this development lies in finding technical solutions for planning that are applicable to real world tasks. The simulation of human planning behavior is scarcely relevant. One exception is the General Problem Solver (GPS) (Newell & Simon, 1961; cited in Inder, 1996), a predecessor of STRIPS. Like STRIPS, it worked with means-ends analysis, but it did not use first-order predicate calculus for the description of states. Instead of that, GPS

[11] In this section (2) the notions of operator and action are used synonymously.

had some other useful features that were partly missing later in STRIPS. The interesting point is that GPS was presented by Newell and Simon (1961) for explaining human behavior in a theorem proving task. As mentioned above, in AI research on planning the reference to human behavior is rare. Most subsequent work on GPS accordingly ignored psychological plausibility and presented it purely as illustrating techniques for allowing a computer program to tackle a wide range of tasks (Inder, 1996).

Beyond classical planning

Since the eighties there has been increasing interest in approaches to planning which go beyond the frame of classical planning (Hertzberg, 1995; Hertzberg, 1996; Inder, 1996). For many fields of application the assumptions of classical planning are too restrictive to build adequate planning systems on their foundation. For example, often there is only incomplete information about the environment available. Even the plan to go into the supermarket to buy toothpaste can fail because of multiple changes to the environment which have not been correctly foreseen. The subject of incomplete information is closely related to the dynamics of the real world. It is very unrealistic that the states of the world can only be changed by the planning actor. Another shortcoming of classical planning affects the consideration of time. Often it is necessary to take the duration of operator execution into account or to pay attention to the effects of overlapping operator execution. Further, there are many other subjects within “non-classical” planning; the following examples are intended to convey an impression to the reader:

- Anytime planner, which are able to deliver a plan at any time: The more time they have for plan preparation, the better the resulting plan is, but some plan is always available from them.
- Hierarchical planning, in which a plan is first built from complex operators on a more abstract level: Only when the abstract plan is finished, will a more detailed elaboration on the level of single actions take place.
- Planning with multiple agents: The agents are simultaneously working on executing the plan; their sequences of actions have to be coordinated with each other.

To complete the picture of “non-classical” approaches to planning within AI, the work of Hayes-Roth and Hayes-Roth (1979) has to be mentioned. Their approach, called “opportunistic planning”, is closely related to cognitive modeling. Accordingly, one of their interests was the simulation of data as it was generated by human subjects. The planning task on which their work was based was quite similar to Plan-A-Day. Subjects had to plan a tour of an imaginary town. The problem was to plan the tour in a way in which as many tasks as possible could be carried out at different places in the town. The artificial system that simulated the human data was an implementation of a kind of blackboard architecture. In this architecture several specialists work together on the planning problem. Each specialist is representative of a specific planning rule. Whenever the current state is favorable for one specialist he adds a contribution to the blackboard. The blackboard is the central part of the system in which the different specialists compete for attention. The one who “shouts” most loudly will be selected for determining

the next action. In this respect the approach is “opportunistic”, because actions are carried out when there is a favorable opportunity.

This approach is interesting because it offers an alternative method for planning which is more realistic in some ways. According to Hayes-Roth and Hayes-Roth (1979) subjects worked mostly with a specific plan, which they then repaired or abandoned as problems came to light. This seems to be a good strategy in a changing and partly unforeseeable environment, and exactly this is achieved by artificial opportunistic planning. Accordingly, in opportunistic planning the stages of plan preparation and execution are closely interlocked in contrast to classical planning. Moreover, when one likes to include the notion of “situated action” into the discussion, its relationship to opportunistic planning is obvious.

2.3 Connectionism and Planning

As it was pointed out at the end of section 2.1.2 concerning connectionism in psychology, high-level cognitive functions like planning or problem solving are mainly modeled by symbolic systems. The author does not know of any connectionist model possessing these functions which can be found within the field of psychology. On the contrary, in AI at least parts of a few models are realized by connectionist networks. For example, in the work of Schmidhuber and Wahnsiedler (1993) the presented problem is to plan a trajectory for an ‘animat’[12] that had to move through an environment with several obstacles. In planning the sequence of movement actions a neural multi-layer network is used to determine the points in the environment where the linear movements of the animat begin and end, respectively. Returning to the blocks world, Bourbakis and Tascillo (1997) present an approach for the coordination of two robotic hands. They represent the planning problem and its states in a special structure, called a stochastic petri net (SPN). A self-organizing neural network for categorizing binary vectors is used to search the SPN structure for an appropriate selection of plans. A very different approach within the blocks world is taken by Ribeiro et al. (1993). They use several very simply structured agents which exchange certain forms of energy with each other. In this respect, their model has some similarity to connectionist models, and for this reason it is presented here. The special feature of the agents is that they are identified with the cubes and the table in the blocks world. Roughly speaking, after each step of propagating energy, the agent with the highest energy is chosen. The cube identified with such an agent is subsequently moved. This local computation allows a close interlinking of plan preparation and plan execution (which leads to a non-classical version of the blocks world).

Arguments in favor of connectionism

All the aforementioned research from AI is not intended to serve as a model of human planning. On the other hand, so far psychology has not used connectionist networks to model planning at

[12] An animat is like a robot that does not exist physically but only as simulated creature in a simulated environment.

all. In spite of these facts, there exist several good reasons why connectionist models are promising even for high-level cognitive functions like planning. First, they have an inherent neurobiological plausibility. The human brain, that is planning and problem solving, is built with neurons. Even if artificial neurons have certain differences and simplifications in comparison to real neurons, the similarity of neural networks to the substrate of human information processing is much stronger than the similarity of symbolic systems like production systems or classical AI planning systems.

Beyond this, connectionism provides an ideal computational architecture for intelligent systems (Shastri, 1991). Shastri writes (1991, p. 261): "Given that intelligent behavior requires dense interactions between many pieces of information it would seem appropriate to treat each memory cell not as a mere repository of information, but rather as an active processing element capable of interacting with other such elements." This argument makes a claim for massive parallel information processing as it takes place in connectionist models. Traditional computational architectures with a single central controller are not appropriate for such a demand.

Given a parallel architecture, it would be useful or even necessary to minimize the costs for communication between the nodes. This is especially true under evolutionary considerations: Because of the pressure of natural selection, the intelligent behavior found in human beings and animals cannot be based on a system that wastes computational power. Communication costs have two components: Encoding/decoding costs and routing costs. The former costs arise from converting information into a format in which it can be transmitted and from reconverting it after it has been received. These costs can be avoided if one uses a kind of information structure that needs no encoding and decoding as is the case in neural networks: Only scalar messages are sent from unit to unit, there is no internal structure at all. Routing costs are necessary to determine the receiver of a message and to establish a communication path between sender and receiver. If one uses fixed connections, these costs can be omitted. Thus, because the connections between units in neural networks are fixed and ready for use at every time, routing costs equal zero in connectionist models.

To summarize, several arguments stated that the features of an appropriate computational architecture for an intelligent knowledge-intensive system are shared by the connectionist approach: Massive parallelism, no central controller, scalar messages with no internal structure, and hard wired links.

Arguments against connectionism

Barnden and Pollack (1991) present several reasons why current connectionist models are not suitable for high-level cognitive functions. Three of their arguments are presented in this section.

First, there is the issue of complex representational structures. This becomes especially clear in the domain of language understanding. Understanding a complex sentence involves constructing a manipulable representation of its content. This principle can be extended to understanding in general. For example, constructing manipulable representations of the environment

is very important for planning. It is difficult to see how such representations are to be embodied by the standard means of connectionist models (feature vectors, weights). The problem is that the system must not rely on surface features of the sentence or the environment, but that a meaningful representation must be extracted from the deep structure no matter how different the surface features may be (e.g., two sentences which are very different in the words used and their order may have got nearly the same meaning).

Second, rapid learning ("insight") is quite difficult for connectionist networks. They usually only perform slow adaptation, which requires thousands of training events. Other forms of learning, which are important for high-level cognition, are beyond their scope. Rapid learning includes learning through instruction, learning by rapid generalization over a small set of instances, analogy-based learning, and explanation-based learning. Especially learning by rapid generalization and by analogy are part of human problem solving and planning capabilities.

Third, "variable binding" or temporary associations are a problem for connectionist models. Variable binding means that certain processing entities ("variables") are linked to specific contents ("values"). For example, in a rule like "if cube x is on top of cube y, cube y cannot be moved" the variables x and y have to be associated with certain cubes in the current state of the blocks world. Otherwise the rule is worthless for the system. These associations have to be built up and to be abandoned in a very short time. Barnden and Pollack (1991, p. 7) write: "It is difficult to allow the binding of values to variables while avoiding an explosion of units and connections, in the more standard sorts of connectionist systems." And further: "Interestingly powerful binding capabilities are, however, provided in less standard systems..." Shastri (1991) presents such a connectionist system that can perform a broad class of deductive inference involving variables and multi-place predicates. The special feature of the system is that not only the amplitude of unit activation is used as information, but that temporal patterns of activity serve as a representation of variable binding.[13] Shastri (1991, p. 277) writes: "Reasoning in the system corresponds to a transient but systematic propagation of rhythmic patterns of activation, where each phase in the rhythmic pattern corresponds to an object involved in the reasoning process and where variable bindings are represented as the in-phase (synchronous) firing of appropriate nodes." Such an approach is clearly different and more advanced than standard connectionist models.

When one summarizes the presented pros and cons for connectionist modeling of high-level cognition, a fairly mixed picture evolves. On the one hand, there are strong points of criticism of standard connectionist models, on the other hand, basic considerations show the plausibility of the connectionist approach in general. In addition, a few more advanced connectionist models are able to overcome some of the stated weaknesses. Thus, it seems to be promising to continue the further development of connectionist models and to apply them to high-level cognition. Our brains are composed of neurons, and it is an urging question which structure and

[13] In this respect, the model presented by Shastri (1991) is much closer to biological neural networks which also work with frequency modulation instead of amplitude modulation. The latter is usually used by standard connectionist models.

which size neural networks must have to carry out tasks like reasoning or planning – tasks we complete very easily using the networks in our brain. We have just set foot into the field of connectionist modeling of human planning, but nevertheless, we should try to continue on further into it. In this respect, this work should serve as a contribution to this undertaking.

3 Introduction to Plan-A-Day

As mentioned in the introduction to this thesis, Plan-A-Day (PAD) is the diagnostic instrument, for which a connectionist model of planning has been developed. In the following I would like to give an introduction to PAD that focuses on the topics which are relevant for this work.

3.1 Plan-A-Day as Diagnostic Instrument

3.1.1 Objective

PAD is a diagnostic instrument for the assessment of planning capabilities of executive personnel, that was recently developed by Funke and Krüger (1995). PAD is completely implemented as a computer application[14] and can easily be modified concerning its demands and difficulty. Because of this it also can be applied to patients with neuropsychological deficits. PAD was developed on the background of a shortage of adequate diagnostic instruments for assessing planning competency. As a special feature, PAD allows not only a performance-oriented evaluation, but also a view onto the process of planning.

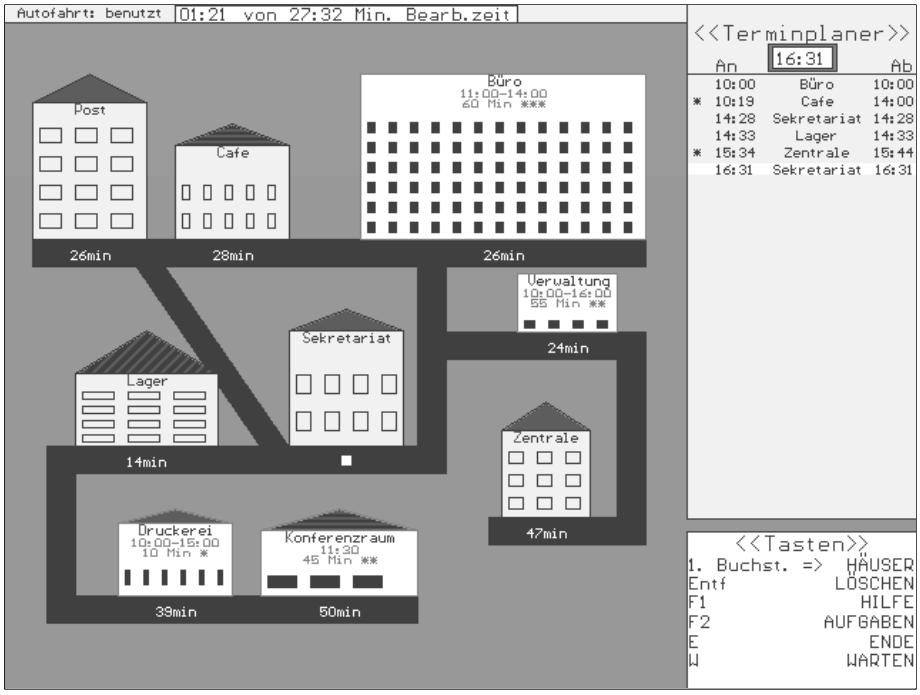

Fig. 3.1: The user interface of PAD as it appears on the computer screen. On the left side one sees the site plan, on the right the list of movements and visits which are already carried out. In the lower right corner a list of available operators is located.

[14] PAD is written in Turbo Pascal 7.0 and runs under MS-DOS or Windows.

3.1.2 Description

The scenario

In PAD, the scenario presented to the subjects is the following: The subject is an employee of a company the buildings of which are scattered over a wide area. The task is to arrange a schedule for the current day, during which several appointments have to be carried out. At the beginning of the day, the subject receives a list of those appointments. Every appointment has several parameters: The earliest and the latest time, at which the task can be started, the task's duration, its priority (normal, high, very high), and its location (in doing so, every of the company's buildings can occur only once in the list of appointments for the current day). The day begins at 10.00 a.m. at the office building and ends some time in the late afternoon. During this period, the subject must carry out as many appointments as possible by planning the sequence of visits to the different buildings in the best possible way. In addition to the earliest and latest time for starting a task, the subject has to consider the time needed for moving from one building to the other. Every move has the corresponding time. Once in a day a car may be used: This reduces the movement time by a third.

Figure 3.1 shows the user interface of PAD. The site plan is located in the left part of the screen. Nine buildings belong to the site (the German notions as they appear in fig. 3.1 are put in parentheses): Post office (Post), café (Café), office (Büro), storehouse (Lager), secretary's office (Sekretariat), administration (Verwaltung), printing office (Druckerei), conference room (Konferenzraum), and central office (Zentrale). In fig. 3.1 the subject is shown as the small white square below the secretary's office. The movement times from the secretary's office to the other locations are written below every building (the complete matrix with movement times is given in appendix B). The earliest and latest starting time, the task duration, and the priority (shown by one to three stars) are displayed within the upper part of each building for the locations for which an unfulfilled appointment exists. The conference room is an exception because the conference always begins at a certain point in time. Buildings with unfulfilled tasks are shown in white with black windows, while the other buildings (either without any task or with a task already carried out) are displayed in light gray.

Furthermore, the message "Autofahrt benutzt" in the upper left corner indicates that the car has already been used. The right part of the screen shows the current state of the planning process in the form of a list of all moves already made. The subject is free to choose any move he wants, even to places where the list of appointments for the current day does not schedule any task. The time shown below the word "Terminplaner" in the upper right corner indicates the current time. Even if PAD can by executed in different modes of presentation which influence the information presented in the user interface, I would like to confine myself to this description of the easiest mode because the following work is also confined to this mode.

Applicable operators and actions

The lower right corner of the user interface (fig. 3.1) shows the commands available to the subject. Besides pressing the ‘F1’ of ‘F2’ key for getting a help screen or the list of appointments to be carried out, there are several commands which are associated with certain actions or operators within the PAD world. In contrast to section 2, with regard to PAD, I would like to differentiate between the notions of operator and action. Theoretical reasons are given in section 3.3. For the present, this differentiation is based only on practical reasons. Within this framework of assignments, actions are a subset of operators. That means, every action is an operator, but not every operator is an action.

First I would like to introduce the possible actions: By pressing the first letter of the German building descriptions one can move to the corresponding place. If one arrives at a location with an unfulfilled task at a time before the earliest time of task execution, one is asked by the system if one chooses to wait until the appointment can be carried out. The “wait” action is invoked by pressing the letter ‘w’ on the keyboard. In addition, if one chooses to drive by car (“drive-by-car” action), one has to press the ‘a’ key before specifying the destination.

A special feature of PAD is the opportunity to delete actions. By pressing the ‘DEL’ key one can undo the previous movement. In this way one may even delete the whole plan and begin again at 10 o’clock in the morning at the office building. Deleting is an operator, not an action.

The last applicable operator is finishing the planning process by pressing the ‘e’ key. After the application of this operator no further planning is possible. The plan that is accomplished so far is interpreted as result by the PAD system.

To sum up, thirteen different operators are available to the subject: Nine *Movements* (according to each of the nine buildings on the site), *Drive-by-car*, *Wait*, *Delete*, and *Finish*. In addition, *Movements*, *Drive-by-car*, and *Wait* belong to the set of available actions (thus, altogether there are eleven actions).

Definition of “PAD task” and “plan” within the PAD domain

In the following the term “PAD task” refers to one day in the PAD world during which a certain set of appointments is to be accomplished. The PAD task is defined by this set and the parameters of the appointments within the set. As there are only nine locations in the PAD world and every location can only be used for one appointment, the maximum number of appointments for one PAD task is nine. (In contrast to “PAD task”, the term “task” refers to the activity demanded by a single appointment.)

The outcome of a PAD task is a plan as a sequence of actions. Within the PAD domain, the term “plan” may correspond to any sequence of actions that is created during the process of planning. However, the sequence of operators (actions plus *Delete* plus *Finish*) which are applied during the process of planning is not named “plan”. To emphasize this even more: The *Delete* operator is not part of the plan but modifies the current plan by undoing actions.

Course of a PAD assessment

When the standard settings for PAD are chosen, the course of a PAD assessment begins with the presentation of the instructions on the computer screen. Next, the subject has to plan his first day in the PAD world. This first PAD task is only intended to be an exercise so that the subject can practice.

Afterwards, the subject has 30 or 40 minutes of time to carry out two PAD tasks which are used to evaluate the planning competency of the subject. After 15 respectively 20 minutes the first PAD task is finished automatically even if the subject has not used the *Finish* operator. If the subject finishes the first PAD task earlier by an explicit *Finish* operator, the saved time may be used for the second PAD task. A warning beep sounds five minutes before the time limit is reached.

Evaluation of a PAD task

The final plan that is accomplished in a PAD task is evaluated according to the priority of the appointments that are successfully carried out in the plan. Each appointment with very high priority counts eight points, each appointment with high priority three points, and each normal appointment one point. The points are summed up, and the result is the end score (or end rating).

In addition, a “max score” (or “max rating”) is calculated, which is also based on the priority of the successfully visited places. The max score is not based on the plan generated directly before finishing. Instead, the max score corresponds with the score for the best plan obtained during the whole planning process.

As there are two PAD tasks in each assessment, four indicators for pure planning performance are calculated for each subject. Beyond this, PAD allows for an evaluation of many characteristics of the planning process. Every keystroke of the subject is recorded into a log file. I will go deeper into this topic when comparing the planning behavior of the connectionist model and of human subjects in section 6.1. In addition, Funke and Krüger (1995) provide a detailed description of the configuration of PAD and its evaluation, especially in respect to the process of planning.

Predefined PAD tasks

Apart from the exercise task there are 16 predefined PAD tasks which differ in several respects. First, their difficulty is different. While the easiest PAD tasks have a set of only four appointments, the most difficult tasks designate an appointment to every location. To increase the difficulty, in some PAD tasks it is not possible to carry out every scheduled appointment.

Furthermore, the predefined PAD tasks differ in their number of rational plans and optimal plans. The former relates to the number of plans in which an appointment by every visit at any location is fulfilled. Plans that include senseless movements are excluded from the set of rational plans. The number of optimal plans, however, comprises every rational plan which leads to the maximum obtainable score (called “optimal score” in the following). Both values

may serve as an indicator for the difficulty of a PAD task. The range for the number of rational plans is between 8 and 47.659, the range for the number of optimal plans between 1 and 4. The maximum score ranges between 19 and 35.

Even if one may choose any two of these PAD tasks for a PAD assessment, there are four recommended pairings which correspond to four predefined levels of difficulty. These pairings are (the following numbers correspond to the predefined PAD tasks): 4 and 5 (easy), 7 and 8 (medium), 13 and 14 (difficult), 15 and 16 (very difficult). While in this thesis only the easiest level was employed, one of the empirical studies presented in section 3.2, for example, made use of all levels.

3.1.3 Special Features of the PAD Conception

Even if most of PAD's features were mentioned in section 3.1.2, it is worth highlighting some of them once more.

First, the semantic outfit of the scenario is closely related to the professional everyday life of executive personnel. This should increase the readiness of such persons to act within the PAD world, as Funke and Krüger (1995) write.

Second, since there are two PAD tasks within each assessment, the reliability of the instrument is enhanced.

Third, within each PAD task the appointments have different priorities, which has to be taken into consideration by the subjects. This differentiates PAD from other instruments in which the priorities are assigned by each subject individually without the experimenter knowing how this is done. For the precise evaluation of a plan, however, it is necessary to know the priorities given to different tasks. Thus, the explicit assignment of priorities in PAD constitutes an important improvement.

Fourth, PAD offers a special aid for planning, namely, the drive by car. This "joker" can be used once, and only by employing it at the right point is one able to carry out certain appointments. Using this special feature of PAD, the experimenter is able to test if such an aid is used by the subject efficiently.

3.2 First Empirical Results

Altogether, Funke and Krüger (1995) present three studies which provide first empirical data regarding PAD.

First study

In the first study, PAD was applied to 104 students of the university of Bonn. This study compares the different levels of difficulty and the different modes of presentation. I will focus on the former comparison. The different levels of difficulty correspond to the pairings of

predefined PAD tasks 4 and 5 (easy), 7 and 8 (medium), 13 and 14 (difficult), and 15 and 16 (very difficult), as mentioned in section 3.1.2. The order of presentation within each pairing never changed.

Table 3.1 shows the mean values for the end score and max score. Since the theoretically attainable maximum scores are different for each PAD task, these values are noted in the table either (in the column "Optimal score"). Because of these differences the end scores cannot be compared directly. The ratio of the end score to the optimal score yields a better means for comparison and can be gathered from the table too (Table: "relative end score"). In the column "Number of operators" one finds the mean number of operators the subjects applied during the planning process. Furthermore, for each PAD task the number of appointments is noted and, in parentheses, the number of appointments which have to be carried out in the optimal plan(s).

As table 3.1 shows, the max scores are usually above the end scores. Thus, subjects often finish the planning process with a plan which is not as good as the best one they designed during the test. The relative end score differs unsystematically between the different levels of difficulty, so that one may question the appropriateness of this classification, if one takes only this indicator into consideration. In addition, for the medium and difficult level, the relative end score shows training effects from the first to the second PAD task. However, because of the fixed order of presentation, this result is confounded with the characteristics of the PAD tasks.

For estimation of reliability, 60 subjects, who were assessed in the easiest mode of presentation, were chosen from the total sample. The correlations between the first and second PAD task of each assessment were calculated. The correlations were .562 for the end score, and .630 for the max score. These correlation coefficients are relatively low, so that further improvements of PAD should bear reliability in mind.

Table 3.1: Mean values for the max score respectively end score compared to the optimal score for each PAD task

Level	PAD task	Number of appointments (optimal)	Optimal score	Max score	End score	$\frac{\text{End score}}{\text{Optimal score}}$ [%]	Number of operators	N
Easy	4	5 (5)	28	26.7	22.7	81.0	16.0	15
	5	6 (6)	26	23.5	20.9	80.5	18.0	15
Medium	7	7 (7)	34	30.8	28.4	83.5	13.0	15
	8	7 (7)	34	30.9	30.9	90.8	13.2	15
Difficult	13	9 (8)	35	30.6	29.4	84.1	23.1	60
	14	9 (8)	35	31.9	31.7	90.4	19.6	60
Very difficult	15	9 (7)	34	30.4	28.6	84.2	17.6	14
	16	9 (7)	34	30.0	28.5	83.8	22.9	14

Second study

In the second study, performed by the "Institut für Wirtschaftspsychologie" (Dortmund), data regarding divergent validity was obtained. PAD was employed in an assessment center in which the subjects (n = 78) were rated according to many different variables: Ability to work in a team, costumer orientation, ability to make decisions, quality of leadership, organizational capabilities, etc. In addition, PAD tasks 13 and 14 were performed.

A factor analysis showed that PAD establishes its own factor (loading: .69), while the other variables load mainly on one other factor. Even if PAD only explains 13% of total variance, this result shows the independence of PAD in respect to other variables.

Third study

The third study (Evers, 1995; cited in Funke & Krüger, 1995) deals with the comparison of executive personnel with control subjects. If PAD is a valid instrument for the assessment of executive personnel, the PAD scores of managers should be better than the PAD scores of control subjects. As the study of Evers shows, this expectation is fulfilled. The mean summed up end scores for PAD tasks 13 and 14 amount to 62.9 for a sample of 22 executives and to 58.1 for a sample of 16 control subjects. This difference is significant ($t = 2.29$, $df = 34$, $p < 0.05$).

Conclusion

As Funke and Krüger (1995) state, PAD is an diagnostic instrument that has reached an intermediate stage of test development. The results for reliability and validity show that further improvements are necessary. Nevertheless, even at the current stage of development PAD is able to serve as an useful instrument in research.

3.3 Theoretical Classification of PAD

Regarding psychology, the question arises, if PAD is more related to problem solving in general or to planning in a narrower sense. The developers (Funke & Krüger, 1995) claim, that their instrument is intended to be a measure for planning capabilities. Certainly, the task for the subjects is to find the optimal sequence of actions within the PAD world, and this clearly is a planning problem. However, on the other side, since the stages of plan preparation and plan execution are totally intermingled within the simulated environment of PAD, a large difference to real planning is the fact that every action may be undone. Normally, one is able to undo actions only in the stage of plan preparation when one is thinking about the plan. However, thinking about carrying out an action does not cause the environment to change as it happens in PAD. Otherwise, in the stage of plan execution, when actions actually have an effect on the environment, actions cannot be undone. At least the lost time cannot be recovered.

Thus, this characteristic of PAD shows its close range to problem solving in a more general sense, where operators may be undone, and where the problem solving process may go back and

forth to every known state in the problem space (see section 2.2.1). Because of this, only the operators referring to real actions within the PAD world (thus, *Movements*, *Drive-by-car*, and *Wait*) are called actions in the following. The *Delete* operator is excluded from the set of actions, because it belongs more to the level of problem solving in general. Moreover, the *Finish* operator is also excluded. Finishing the planning process is not an action carried out within the PAD world, but more of a meta-cognitive operation.

Interestingly, PAD resembles the paradigm of classical planning within AI. If one compares the ten preconditions of classical planning stated in section 2.2.2 with the characteristics of PAD, many of these preconditions are fulfilled in the PAD world. Thus, PAD provides a static environment with exactly defined states which are only subject to change by the actions of the planning agent. However, there are also some differences between PAD and classical planning. First of all, time is an explicit part of the PAD world and has to be taken into consideration. Furthermore, the stages of plan preparation and execution are not separated in PAD. As pointed out before, they are intermingled in a very special fashion, because on the one hand, the environment is changed by actions (like during execution), but on the other hand, every action can be undone (like during preparation).

Precondition (viii) states that the objectives for the resulting plan are explicitly given. In PAD, only the vague objective to carry out as many appointments as possible under consideration of their priorities exists. However, which subset of the scheduled appointments forms the optimal plan(s) has to be found out by the planning agent. In this respect, preconditions (viii) and (ix) (“A ready-made plan has to achieve all its objectives”) are not fully appropriate for PAD. Precondition (x) regarding the computing time is not relevant for PAD.

In summary, the place of PAD within the theoretical framework is on one hand, close to problem solving in general, on the other hand, some of PAD’s characteristics show that it belongs to the field of planning even almost in the classical sense as it is defined within AI.

4 A Connectionist Model for Plan-A-Day

This section provides a detailed description of the connectionist model which was developed for Plan-A-Day. The straightforward presentation given does not reflect the real course of development. The model, as it is presented here was created in a continuous feedback process between all stages of model development and testing, including the stage of fitting the model to empirical data. The procedure of model fitting is presented in section 5.3.

4.1 Foundations

The objective of modeling was to find a model which is able to simulate human planning behavior in PAD tasks. In doing so, it was not intended to reproduce data generated by individual subjects. Even if the model simulates singular subjects, the comparison takes place on the level of samples – on the one hand, a sample of human subjects, on the other hand, a sample of simulated subjects.

There are several possibilities to simulate a subject. One may restrict this simulation to the generation of the resulting plan, or one may model the whole planning process step by step. In this work the second approach is taken. Thus, theoretically, the model could be seated in front of the computer screen to work on a PAD task instead of a real human subject. Like a human subject, the model generates a sequence of operators in order to find the optimal plan. As explained later in detail, in each processing cycle the model evaluates the applicable operators. Therefore, the model is named "EVA" – an abbreviation for "Evaluation of Actions". More precisely, it should be called "Evaluation of Operators", but "EVA" sounds better than "EVO".

Every operator EVA applies changes the state of the PAD world as is the case for a human subject. According to the new state EVA produces the next operator. Thus, EVA's input is the current state of the PAD world including the current PAD task, and its output is the next operator. This approach is closely related to situated action respectively situated cognition. As Clark (1997, p. 60) writes: "Connectionist minds are ideal candidates for extensive external scaffolding." As pointed out in section 2.2.1, external scaffolding means leaning on the environment in problem solving or planning. Since connectionist models are mainly capable of pattern completion or transformation, the environment is a quite valuable source of information for them. Rumelhart, Smolensky, et al. (1986) explain by the means of long multiplications, how this interplay works. Simple multiplications, such as 5 x 7 = 35, can be supported by pattern-recognition devices. Long multiplications like 3254 x 5647 are more difficult, but by using external help this problem can be divided into many simple multiplication and addition steps. Human beings usually use paper and pencil for this purpose. These simple operations are then again a pure task of pattern completion. To cite Clark (1997, p. 61): "... by an interrelated series of simple pattern completions coupled with external storage we finally arrive at a solution." Clark (1997) uses this example of Rumelhart, Smolensky, et al. (1986) to point out,

that pure pattern-completing abilities combined with a complex, well-structured environment may enable “connectionist minds” to fulfill complex tasks in general.

Thus, on the one hand, EVA relies on the approach of situated cognition[15]. Further, EVA has been influenced by MEKIV (Hussy, 1993). MEKIV is a model for complex human information processing, which provides a framework for didactics and research. In spite of the fact that MEKIV is not very well suited for connectionist modeling since it includes a kind of central processing unit, at least one idea was taken from this model. MEKIV states that in problem solving operators and evaluators are retrieved from memory. Evaluators are used for evaluating the current state and for evaluating operators in regard to certain sub-goals. EVA is based on this principle. Even if EVA is at first only oriented toward the main goal of achieving the optimal plan, EVA evaluates in every processing cycle every applicable operator in accordance with this goal.

To sum up, EVA is intended to be a connectionist model generating several evaluators (evaluations of operators[16]) simultaneously according to the current state of the environment. Which kind of network model may be used to perform such a task? As it was pointed out in section 2.3, standard models of connectionism are not very well suited to fulfill high level cognitive tasks such as planning. Problems arise regarding among other things, complex representations, variable binding, and rapid learning. However, on the other hand, standard models are already quite well understood and, in comparison, quite easy to implement. Further, it would be a strong argument in favor of connectionism if one could avoid the stated problems and apply a standard model to planning,. This would show that even simple pattern recognition capabilities are enough for explaining human planning behavior in a specific domain.

For these reasons a traditional model of connectionism was chosen for EVA: A multi-layer feed-forward network with logistic activation functions (in the following simply called multi-layer network[17]) as described in section 2.1.1.3. In contrast to other standard models it has several useful properties:

- Multi-layer networks can perform complex pattern transformations. As the proof of Hornik et al. (1989) shows, nearly every transformation function can be represented by a multi-layer network of appropriate size. In contrast, e.g., self organizing maps (Kohonen, 1982) are only capable of classifying patterns.
- While the input and output vectors may be held constant, the performance of a multi-layer network can be improved by enlarging the hidden structures. In contrast, in other network models the representation of the input/output (at least the size of the input/output vector) specifies the whole network topology. In addition, such models like the Hopfield network (Hopfield, 1982) or the bi-directional associative memory (Kosko, 1987) are restricted to a

[15] The approaches of situated cognition and situated action are so closely related to each other that I won’t always state both terms explicitly, even if both terms are meant.

[16] In the work of Hussy (1993) the meaning of “evaluator” is slightly different. For him, an evaluator is an evaluating process that is applicable to any state or operator. However, for the purpose of my work it is more useful to define an evaluator as an evaluation of a single operator with regard to a certain goal.

[17] In respect to the fact that EVA uses additional recurrent connections (which is explained later), the notion "multi-layer network" is also more appropriate.

certain number of differentiable patterns due to their number of units. Thus, in such models there is an interaction between the size of the input/output vector and the information storage capacity. This restriction is not acceptable for EVA, because EVA should be able to work with a large set of different PAD tasks which constitute a set of input vectors which is just as large.

- For multi-layer networks with logistic activation functions the number of training patterns is not confounded with the network topology as is the case for radial basis functions networks or probabilistic neural networks (Zell, 1997). In these models, the number of units in the first hidden layer is equal to the number of training patterns. For EVA, with the broad variety of the PAD world, this would lead to a network of immense size.
- Multi-layer networks work with continuous input and output values. As it will become obvious in the following subsections, EVA relies heavily on this feature.
- Even if the learning procedures for multi-layer networks are not plausible in respect to biology, at least their topology and the kind of units employed are closer to biological reality than is the case for many other connectionist models.

4.2 EVA in More Detail

4.2.1 Overview

In this section I would like to present EVA in more detail. In determining the concrete specifications of EVA a characteristic problem of connectionist modeling arises. On the one hand, one would like such a network model to have interesting emergent properties. For that reason, one avoids giving the model a certain high-level functional structure in advance. However, on the other hand, a skill such as planning, presumably needs the coordinated work of several sub-modules. These claims contradict each other, because the second way diminishes the possibility of emergent features, and the first way simplifies the inner process of planning in a quite strong fashion.

Finally, as it will become evident in the following sections, EVA is a compromise between both ways. As fig. 4.1 shows, EVA consists of sub-modules, but these sub-modules work parallel in the same functional context. Before I address the question of EVA's internal structure more closely, I must first give an overall view of the model. According to fig. 4.1, in every

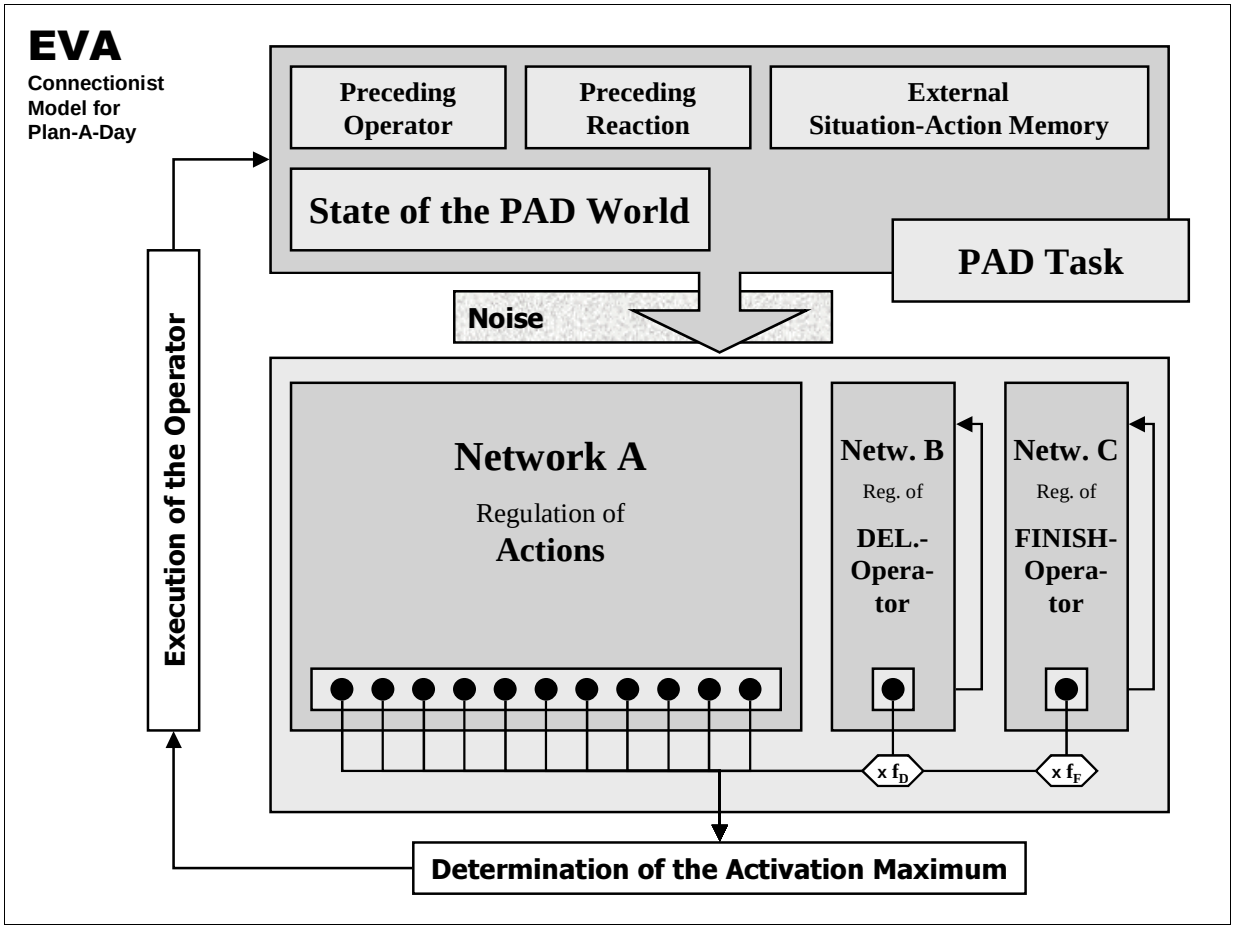

Fig. 4.1: Basic architecture of EVA. In the upper part the structure of the input is shown, underneath the actual model composed of three sub-networks. The output units are displayed as black dots. By determination of the activation maximum within the output units the next operator is chosen. Its execution alters the current state of the PAD world and with that the input (following the feedback arrows).

processing cycle EVA receives an input composed of several components. This input is applied to EVA's sub-networks. After the input is propagated through these multi-layer networks, EVA determines by the activation of the output units, which operator is carried out. Through the execution of this operator the state of the PAD world and with that the input for the next processing cycle, is changed. This cyclic process goes on until EVA applies the *Finish* operator.

Table 4.1: Composition of the PAD task

For each of the nine locations:
- Flag, whether there is a scheduled and unfulfilled appointment

→ If there is a scheduled appointment:
- Priority of the appointment
- Earliest starting time of the task
- Latest starting time of the task
- Duration of the task

Table 4.2: Composition of the state of the PAD world

For each of the nine locations:
- Flag, whether the appointment (if scheduled) was already successfully carried out
- Flag, whether there was a visit without carrying out an appointment
- Flag, whether the location is identical with the current location
- Time needed to reach the location from the current location

General information about the current state:
- Current time
- Flag, whether the drive by car was already used

4.2.2 The Output

In explaining EVA it is easiest to begin with the output. As was pointed out in section 3.1.2, 13 operators are applicable within the PAD world: Nine *Movements*, *Drive-by-car*, *Wait*, *Delete*, and *Finish*. Correspondingly, EVA has 13 output units altogether (displayed as black dots in fig. 4.1.). Every output unit is assigned to one operator. Since the output units act as evaluators, the activation of each output unit represents the evaluation of the corresponding operator in the current context. In every processing cycle the operator with the best evaluation, also referred to as the output unit with maximum activation, is determined. The corresponding operator is carried out afterwards.

4.2.3 The Input

Environment

The input to EVA consists of several components. First, there is the PAD task itself (in detail in table 4.1). Because the PAD task remains constant for the most part through the whole planning process, it is placed across the bottom right corner of the input box in fig. 4.1. The other components, however, are subject to change by nearly every operator execution. The state of the PAD world (in detail in table 4.2) is influenced by operator execution, and the preceding operator and reaction are influenced as well. The preceding operator is the operator applied by

EVA as result of the previous processing cycle. The preceding reaction, in analogy, is the direct reaction of the PAD system to this operator. This reaction depends on whether the operator is a valid user input or not. The exact assignment of reactions to operators is shown in table 4.3. So far, the input corresponds closely to the information visible for the human subject in the PAD user interface, so that human subjects and EVA have very similar information available for their "situated actions". Of course, the transfer of the user interface shown in fig. 3.1 to a vector of scalar values (see section 4.3.1) cannot be made without certain distortions. For example, sometimes preceding operator and reaction are displayed by the real PAD system in a very emphasized matter, and sometimes they have to be searched for in different places in the user interface and must then be completed with information from short-term memory.

External situation-action memory

An external situation-action memory is also provided in the input. This memory serves as a supplementary aid for EVA. For every situation (that means, for every possible state of the PAD world) it stores, which operators were applied in this particular situation up until the present point. Furthermore, every new storage for an already encountered situation weakens the prior existing traces for this situation. Such a memory system does not claim to be more than a loose analogy to existing memory models in psychology (e.g., Hussy, 1993). On the one hand, the situation-action memory corresponds to the episodic long-term memory (with memory traces that have just gone from short-term to long-term memory), on the other hand, the weakening of prior existing traces resembles the process of interference which is especially strong during the stage of consolidation and for traces which are similar to each other (Zimbardo, 1995).

During the development of EVA, it became clear, that such a memory aid is needed because not even the recurrent version of multi-layer networks is capable of providing a recollection of the previous course of planning as well as is necessary. It would be nice to implement the situation-action memory by another connectionist model (e.g., by a bi-directional associative

Table 4.3: Conditions for negative reactions of the PAD system regarding the applicable operators. Under all other conditions within the PAD world the reaction is positive.

Type of operator	Reaction: negative (operator rejected)
Movements	• When the arrival at the corresponding location would be after 6.30 p.m. • When the subject is already at the location.
Drive-by-car	• When the drive by car already has been used.
Wait	• When the *Drive-by-car* operator has been invoked immediately before. • When there is not any appointment to be carried out at the present location. • When the earliest task starting time at the present location already has passed by.
Delete	• When there is nothing to delete (the current plan is empty).
Finish	May always be applied.

memory, BAM [Kosko, 1987]), but unfortunately the available amount of time was not enough for this undertaking. Thus, the external memory is implemented by a conventional procedure within the PAD simulation system.

Noise

In the framework of EVA, the input is directly fed into the operator-generating system. That omits many intermediate stages of perception and of percept-processing during which the information from the environment is subject to distortion. Noise is added to the input in order to model these distorting influences.

Further, EVA relies on the assumption that the complete input described up until the present time can be held in an unified internal representation at once. Even if this may be correct on a sub-conscious level, for controlled and intentional information-processing this surely is not true. The capacities of short-term and working memory are too small to hold such a complex representation. Considering this, a distortion of the input representation is also meaningful. A detailed description of how noise is applied to EVA's input is given in section 5.3.2.

4.2.4 Internal Structure of EVA

EVA consists of three sub-networks each of which is a multi-layer network. In the following they are called Network A to C, or EVA-a, EVA-b, and EVA-c. Network A is the largest one. It has eleven output units which are assigned to the eleven possible actions in the PAD world: Nine *Movements*, *Drive-by-car*, and *Wait*.

The one and only output unit of network B is assigned to the *Delete* operator, and the one and only output unit of Network C is assigned to the *Finish* operator. As displayed in fig. 4.1, the output of EVA-b and EVA-c is each weighed by a factor (f_D and f_F, respectively), before the activation maximum is determined. This feature is useful for fitting the model to empirical data. Therefore, it is described more detailed in section 5.3.2.

EVA-b and EVA-c have explicit recurrent connections, what is shown by small feedback arrows in fig. 4.1. They are hierarchical Elman networks (Elman, 1990; Zell, 1997). A more precise elaboration is presented in section 4.4.

The subdivision of EVA into several sub-networks has become necessary during model development in order to optimize the training procedure for each operator. This is the technical side. Further, as deleting and finishing are somehow different from generating actions (see section 3.3) and probably more related to meta-cognitive processes (concerning the regulation of the planning process), it may be argued that the sub-modules responsible for deleting and finishing are in fact based on their own sets of neurons. But, of course, on the current level of knowledge, this argument is as fuzzy as the whole topic itself.

At least EVA is not the first connectionist model using several sub-modules. For example, Zell (1997) cites the work of Jacobs et. al. (1991), who used a combination of several local expert networks which were integrated by a gating network (adaptive mixture of local experts).

Table 4.4: Specification of the input vector for EVA-a, EVA-b, EVA-c. For Explanation see section 4.3.1.

Component	Unit	Included in sub-network(s)	Range	Noise factor	Meaning of unit values
PAD task					
For each of the nine locations (9×):					
	State	ABC	0.0 | 1.0	0.05	0.0 ⇐ No appointment at this location (⇒ Priority, Time1, Time2, Duration = 0.0) or Scheduled appointment is already fulfilled 1.0 ⇐ Unfulfilled appointment at this location
	Priority	ABC	0.0 | 0.1 | 0.55 | 1.0	0.025	0.1 ⇐ Normal priority | 0.55 ⇐ High priority | 1.0 ⇐ Very high priority
	Time1	AB	0.0 | 0.1 – 1.0	0.01	Earliest time for task starting: From 10.00 a.m. to 6.30 p.m.(in steps of 5 min.)
	Time2	AB	0.0 | 0.1 – 1.0	0.01	Latest time for task starting: From 10.00 a.m. to 6.30 p.m. (in steps of 5 min.)
	Duration	AB	0.0 | 0.1 – 1.0	0.01	Task duration: From 5 min. to 90 min. (in steps of 5 min.)
State of the PAD world					
General information (1×):					
	CurrentTime	AB	0.1 – 1.0	0.01	Current time: On a continuum from 10.00 a.m. to 6.30 p.m. (later moments are set to 1.0)
	CarUsed	AB	0.0 | 1.0	0.05	0.0 ⇐ Drive by car has <u>not</u> been used yet | 1.0 ⇐ Drive by car already used
For each of the nine locations (9×):					
	VisitCarryOut	ABC	0.0 | 1.0	0.05	0.0 ⇐ Location has not yet been visited or only without carrying out the appointment 1.0 ⇐ Location has been visited with carrying out the appointment
	VisitNonCarryOut	ABC	0.0 | 1.0	0.05	0.0 ⇐ Location has not been visited yet or only with carrying out the appointment 1.0 ⇐ Location has been visited at least once without carrying out the appointment
	CurrentLocation	AB	0.0 | 1.0	0.05	0.0 ⇐ Location is the current location 1.0 ⇐ Location is not the current location
	MoveTime	AB	0.0 – 1.0	0.01	Time needed for reaching the location from the current location: On a continuum from 0 min. to 97 min.

Table 4.4 (cont.)

Component	Unit	Included in sub-network(s)	Range	Noise factor	Meaning of unit values
Preceding operator					
For each operator excl. "Finish" (12×):					
	OperatorFlag	AB	0.0 | 1.0	0.05	0.0 ⇐ Operator has not been carried out in the preceding cycle 1.0 ⇐ Operator has been carried out in the preceding cycle
Preceding reaction					
(2×):					
	ReactionFlag	AB	0.0 | 1.0	0.05	0.0 ⇐ Preceding operator has been a valid user input 1.0 ⇐ Preceding operator has not been a valid user input
External situation-action memory					
For each action (11×):					
	MemoryUnit	A	0.0 | 0.7^x	0.05	For every action there exists a memory unit, which indicates whether the action has been already carried out in the current state of the PAD world. When an action is "remembered", the value of the corresponding <MemoryUnit> is 0.7 [= MemForgetFactor] raised to the power of a whole number, beginning with $0.7^0 = 1.0$. The earlier the corresponding action has been carried out, the larger the exponent is (to model forgetting). The <MemoryUnit> of a not remembered action is set to 0.0.

4.3 Precise Specification of Input and Output

After section 4.2 provided a first glimpse on the overall structure of EVA, this section gives a detailed specification of the input vectors used for the sub-networks of EVA and also a detailed specification of the principles according to whom the output units are intended to work.

4.3.1 Input Specification

Since EVA consists of multi-layer networks each of which has an input layer, the input has to be specified as a vector of scalar values within the range of the logistic activation function, thus within the interval [0.0; 1.0]. Table 4.4 shows all input units and their meaning. The units are listed in the second column, the third column indicates in which sub-networks each unit is used. The input vector of EVA-a comprises all units listed in table 4.4; for EVA-b the external situation-action memory is left out; and EVA-c only comprises a small subset of units related to the state of the scheduled appointments and their priority.

The fourth column of table 4.4 displays the values each unit can adopt. The unit may adopt either several discrete values or any value on a continuum. In the fifth column one finds the "noise factor". This factor determines how heavy noise may distort each unit. The smaller this factor is, the smaller the maximum distortions are. The values of the noise factor are chosen according to the information density of each unit. For example, the information density of a unit like <Time1> is comparatively high, because the range from 0.1 to 1.0 represents a period of 8½ hours. In contrast, the unit <State> can only adopt two discrete values, 0.0 and 1.0. Accordingly, the noise factor for <Time1> is 0.01, and the noise factor for <State> amounts to 0.05.

PAD task

Altogether, there are 108 input units. First, there are 45 units for the PAD task. The PAD task is determined by the appointment parameters for the nine locations within the PAD world. Five parameters constitute each appointment. <State> indicates, whether there is a scheduled appointment at the corresponding location. A value of 0.0 has two possible meanings: Either there is no appointment scheduled at all, and then the other four units (<Priority>, <Time1>, <Time2>, <Duration>) are set to 0.0 as well, or there was a scheduled appointment that has been already carried out. A scheduled and unfulfilled appointment is indicated by a <State> value of 1.0. If there is scheduled appointment at the corresponding location, the other four units specify it more precisely. <Priority> holds the priority of the appointment on three levels: Normal, high, and very high. <Time1> and <Time2> represent the earliest and the latest starting time respectively, of the task on a continuum from 0.1 to 1.0 which corresponds to the period from 10.00 a.m. to 6.30 p.m. (in steps of five minutes). The conversion between both scales is linear. <Duration> holds the task duration, which ranges between five min. and 90

min. (also in steps of five minutes). The conversion to the unit values in the range between 0.1 and 1.0 is linear as well.

The information provided in the PAD task component of the input vector corresponds closely to the information shown in the user interface of PAD (within the roofs of the buildings as shown in fig. 3.1). After carrying out an appointment, the corresponding building is displayed in gray color indicating the state. The main difference is, that after carrying out an appointment the specifications of the appointment are no longer shown in the roof of the corresponding building in contrast to the input vector, where the assigned units still hold their values. On the other hand, the specifications of the PAD task are still completely available to the human subject by pressing the 'F2' key. Thus, the format chosen for the PAD task component of the input vector is a compromise.

State of the PAD world

The second component of the input vector is the state of the PAD world or, to put it more simply, the situation. The situation comprises 38 units, two of which are related to general information: <CurrentTime> and <CarUsed>. <CurrentTime> represents the current time as given in the PAD world. The continuum from 10.00 a.m. to 6.30 p.m. is transformed by a linear function to the range from 0.1 to 1.0. For the rare cases when the current time is later than 6.30 p.m., <CurrentTime> also takes a value of 1.0. <CarUsed> indicates whether the car has already been driven. This unit works like a flag with two states, 0.0 for false ("No, not used yet") and 1.0 for true ("Yes, already used"). Both <CurrentTime> and <CarUsed> are visible in an analogous form in the user interface of PAD: In the upper right and upper left corners of the screen, respectively (see fig. 3.1).

Further, situational information is available for every location as well. For each of the nine locations four units are reserved. Three of them, <VisitCarryOut>, <VisitNonCarryOut>, and <CurrentLocation>, work like flags. <VisitCarryOut> is set to 1.0, when the location has been visited with carrying out the appointment (of course only if an appointment is scheduled). In contrast, <VisitNonCarryOut> is set to 1.0, when the location has been visited without carrying out any appointment, no matter, if there is any scheduled appointment at all, or if the location has been visited at the wrong time. These two flags work as a substitute for the precise list of movements and visits completed so far, as shown on the right side of the user interface (see fig. 3.1). On the one hand, the list provides more information, especially about the chronological course, on the other hand, this combination of flags states explicitly what a human subject has to derive from the PAD task, the condition of the buildings, and the course of movements and visits. This derivation is not difficult, but also not totally obvious. Unfortunately, such an open list cannot be represented within the input vector without providing a large amount of units. Therefore, the combination of <VisitCarryOut> (nearly equivalent with <State>, but reversely defined) and <VisitNonCarryOut> was chosen as substitute.

<CurrentLocation> and <MoveTime> are less problematic. They can be read out directly from the user interface. <CurrentLocation> is set to 1.0, when the location is identical with the current location where the "subject" is at the current moment. Therefore <CurrentLocation>

can only be set to 1.0 for one location at a time. The <MoveTime> unit displays the time it would take to move from the current location to the repective location to which the <MoveTime> unit belongs. The movement times range from zero min. to 97 min.; they are linearly transformed on the range from 0.0 to 1.0. When the drive by car is invoked, all <MoveTime> units are reduced to a third.

Preceding operator

The preceding operator only takes twelve units from the input vector, one for each operator excluding *Finish*. Each unit works like a flag (called <OperatorFlag>): If the corresponding operator has been carried out by EVA in the preceding cycle, the unit is set to 1.0. Therefore, only one unit within the preceding operator section can be set to 1.0 at any time. The unit for the *Finish* operator is left out, because the process of planning always comes to an end after the use of *Finish*.

Preceding reaction

Both units in this section of the input vector work in the same way. The information provided by the <ReactionFlag> is doubled in the input vector, which serves to emphasize this information a little more. When the preceding operator has been a valid user input to the PAD system, the <ReactionFlag> is set to 0.0, otherwise it is set to 1.0. The conditions for positive vs. negative reactions of the PAD system are shown in table 4.3.

The special relation of the preceding operator and the preceding reaction to the information displayed in the user interface is addressed in section 4.2.3.

External situation-action memory

The role of the external situation-action memory within the input of EVA is discussed in sections 4.2.3 and 4.5.2.2. As it is pointed out there, this memory system associates states of the PAD world with actions previously applied to these states. For each of the eleven available actions there is one <MemoryUnit>. The procedure underlying the memory stores for every distinct situation within the PAD world, every action ever applied and their order. When a situation is entered repeatedly, the previously applied actions are represented by the <MemoryUnits>. A <MemoryUnit> for an action never applied before is set to 0.0. The other <MemoryUnits> display MemForgetFactor (a parameter in the interval]0.0; 1.0[), which is raised to the power of a whole exponent. The exponent depends on the order in which the actions were applied. Zero is assigned to the most recent action, one is assigned to the second to last action, and so on. Thus, the <MemoryUnit> for the most recent action is set to 1.0; for the other actions a kind of forgetting takes place. The longer ago they were applied to the situation, the smaller the value of their corresponding <MemoryUnit> is.

MemForgetFactor is one of the explicit parameters of EVA. For the generation of the training data (see section 4.5.2.2) MemForgetFactor was set to 0.7 (as it is shown in table 4.4).

However, one supplemental remark is necessary: The <MemoryUnit> for the *Wait* action only works for the starting situation of PAD (10.00 a.m. at the office building). In this situation *Wait* is like a movement to the office, in order to carry out the appointment scheduled there. In all other situations it would be senseless to provide a memory for waiting, because waiting should always be carried out whenever the planning agent arrives too early at a location, regardless of the existence of a memory for preceding *Wait* actions in this specific situation.

General remarks

In the specification of the full input vector many modeling decisions have been made and many parameters have been set more or less implicitly. Both theoretical and practical considerations have led to the chosen input representation. Guided by the idea of situated action, the main goal was to transform the user interface of PAD into the best possible representation that can be provided by a vector of scalar values between 0.0 and 1.0. Even if the construction of flags seems to be plausible in psychological respect, many other specifications for certain input units are more open to criticism. Flags are used in the form of cues in many other psychological models (e.g., in the BIAS model of Fiedler, 1996), but the representation of time on an activation continuum does not have any previous examples known by the author. Many other representations of time in such an input vector are imaginable too; for example one may use a temperature coding (Blankenberger, 1992). However, the fact that the number of units in the input vector is limited due to calculating capacity, should by no means be overlooked. The one unit coding of time is very economical (even if it is theoretically questionable), and even with this coding, the calculating demands of training EVA exceeds the capacity of a state of the art personal computer (see section 4.5).

Many parameters are implicitly set by determining the value range of the input units. However, since the value range for every output unit follows the general maxim to use the available range of the logistic activation function from 0.0 to 1.0 most extensively, and since the value range was not varied in the process of model fitting (see section 5.3.1), these implicit parameters are viewed in the following as an integral constituent of EVA and not as free parameters.

So far, four “explicit” model parameters have been introduced. First, there is MemForgetFactor. As it will become obvious in section 4.5.2.2, MemForgetFactor is important for generating the training data. Beyond it, MemForgetFactor may be varied for different simulation runs.

The other three model parameters have only briefly been mentioned so far. They can be found in the column “noise factor” of table 4.4. This factor determines, how much the corresponding input unit can be influenced by noise (see section 5.3.2). An inspection of table 4.4 reveals that three different values are given as noise factors: 0.01, 0.025, and 0.05. They correspond to the parameters NoiseTime, NoisePriority, and NoiseFlag. NoiseTime is applied to every input unit dealing with time with a comparatively high information density. NoiseFlag is used for input units working like false/true-flags and for the <MemoryUnits>. The input unit

<Priority> is an isolated case with a fourfold gradation. Therefore, NoisePriority receives a medium value between the other two noise parameters.

4.3.2 Output Specification

4.3.2.1 Output as Evaluation of Operators

As pointed out in section 4.1, in every processing cycle EVA is intended to generate evaluators for each operator. These evaluators are evaluations of operators with respect to the main goal of finding the optimal plan. As stated earlier, every output unit of EVA is assigned to one operator. Thus, considered together, every output unit generates the evaluation for its specific operator. The following describes how this evaluation takes place. This description is correct for the training data, by which both input and output are prescribed by the "external teacher". For the output EVA actually produces these instructions have got only normative character. The better EVA has learned to work on PAD tasks and to produce such evaluators as output, the smaller the difference between the desired output and the actual output of EVA will be (see section 4.5.1). In addition, in section 4.5.2 some further modifications of the (desired) output, which were created due to special demands of single operators are presented.

The basic principle of the desired evaluation is quite simple. First, given a specific PAD task, one determines the score for its optimal plan. Following the procedure described in section 3.1.2, every very important appointment is worth eight points, every important appointment three points, and every normal appointment one point. Additionally, for every location visited without carrying out an appointment five points are substracted (of course, such visits are not part of an optimal plan, but these "penalty points" are needed for the rating of hypothetical plans as described in the following). While the normal evaluation of plans is called "standard evaluation" in the following, the evaluation including penalty points is called "advanced evaluation".

Furthermore, one considers the current situation and the actions carried out so far in the current plan. According to the advanced evaluation, a maximum end score is calculated for every operator, that could potentially be reached, if that operator were to be applied to the current situation, and if the *Delete* operator could not be applied afterwards. This constraint is important because without it, one would always be able to reach the optimal score. Thus, every operator is examined in relation to the maximum end score it allows, if one follows the optimal way beginning with the operator itself, starting from the current situation. For example, for the *Finish* operator this maximum end score is always identical with the score for the plan produced thus far.

The evaluation value for each operator is calculated as the ratio between the operator's specific maximum end score and the optimal score of the PAD task. Therefore, every operator leading to the optimal plan receives an evaluation value of 1.0. Operators leading to inferior solutions receive evaluation values less than 1.0. Unfortunately, this procedure does not work

appropriately for the *Delete* operator. Thus, the *Delete* operator is evaluated according to a different rule: Whenever the current plan does not allow the optimal score to be reached through the application of one or several actions, then the *Delete* operator receives an evaluation value of 1.0. Otherwise it receives 0.0.

This method of evaluation has several implications. For example: Because senseless visits are punished by a deduction of five points, plans including such visits always result fact that the *Delete* operator is evaluated with 1.0 and all other operators with a lesser value.

In addition, two further rules are used in the evaluation of operators. First, actions that are senseless to apply are evaluated with 0.0. *Movements* are senseless in this respect, when no appointment is scheduled at the destination of movement or the scheduled appointment has already been carried out. Further, every action is seen as senseless, if it produces a negative reaction by the PAD system (see table 4.3). The second additional rule refers to the *Drive-by-car* action. Whenever *Drive-by-car* gets an evaluation value of 1.0, but another operator also gets this optimal value, then the evaluation value of *Drive-by-car* is reduced to 0.9.[18] This rule prevents the use of the drive by car in cases, when it is not totally necessary. This is a kind of economical principle.

Following this method of evaluating the operators every output unit receives a value between 0.0 and 1.0, which corresponds to the evaluation value of its assigned operator. This vector of evaluators depends on the given PAD task and on the current state of the PAD world. The method of evaluation reflects the goal of reaching one of the optimal plans – only the operators leading to one of these plans get the maximum evaluation value of 1.0. The other operators get evaluation values corresponding to the best plan to which their application would lead.

4.3.2.2 Accentuation of the Best Evaluators

In addition to the evaluation principles considered so far, one more step has to be carried out to produce the final vector of desired output values. Namely, all evaluation values smaller than 1.0 are multiplied by a restraining factor in the interval [0.0; 1.0]. This factor corresponds to the model parameter RestrainFactor, which is set to 0.4 for the training data of all sub-networks of EVA. A RestrainFactor of 1.0 would omit the accentuation of the best evaluators, a RestrainFactor of 0.0 would lead to output vectors in which the units assigned to operators leading to optimal solutions are set to 1.0 and all others to 0.0.

The introduction of the RestrainFactor leads to a stronger accentuation of the best evaluators because all other evaluators are restrained. During the development of EVA this additional procedure has caused better results for network training and for the performance of EVA on various PAD tasks. Therefore it was included in the model.

In the course of determining the evaluators for a given PAD task and a given situation several implicit parameters find their way into the model again. However, as for the input, a similar argument is stated against viewing at these parameters as free model parameters: The whole

[18] When the competing operator is the *Finish* operator, then the evaluator of *Drive-by-car* is even reduced to 0.0.

evaluation procedure is guided by the main goal to find evaluators which are orientated by the optimal plan(s). Except for RestrainFactor, most implicit parameters are set due to technical or theoretical considerations and were not varied in the process of model fitting.

4.4 Precise Specification of the Sub-Networks

4.4.1 EVA-a

As stated previously, all sub-networks of EVA are multi-layer networks. EVA-a is a pure feed-forward network with four hidden layers. Fig. 4.2 shows the structure of EVA-a. In addition to the connections from each layer to its following layer, there are several "shortcut connections". EVA-a is a fully connected network: Every layer receives input from all of its preceding layers. These shortcut connections were introduced into EVA-a (and also into EVA-b and EVA-c), because they provide additional power. Zell (1997, p. 422) outlines the two-spiral-problem which is much more easy to solve for backpropagation networks, if one uses shortcut connections. Further, Zell (1997, p. 418) recommends the use of shortcut connections in general.

Own pre-experiments with a reduced and simplified version of PAD ("Mini-Plan-A-Day" or MPAD) gave converging evidence. The most important factor that determines the time needed for each training epoch is the number of weights within the network. In keeping the number of weights nearly constant, one can create networks with a different number of hidden units. In

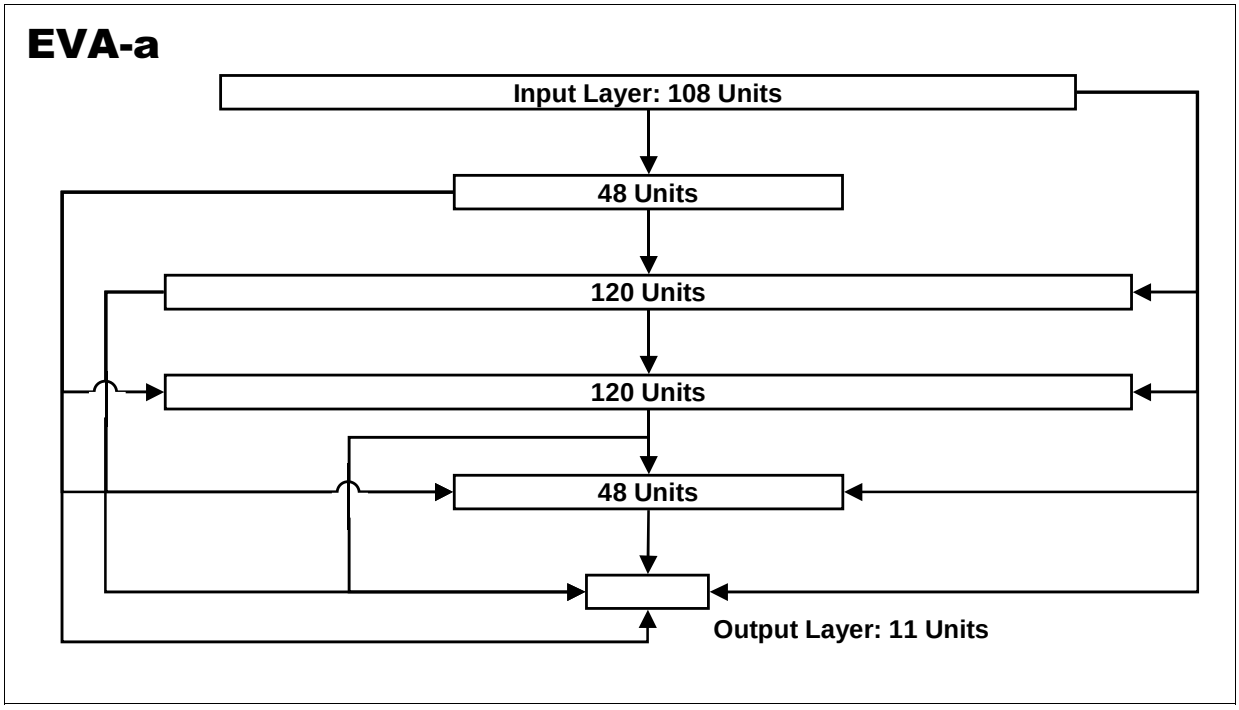

Fig. 4.2: Structure of EVA-a. The unit-to-unit connections are replaced by arrows indicating the connections between entire layers. EVA-a is a fully connected multi-layer feed-forward network. It has 347 active units in its hidden layers and output layer. The number of weights amounts to 81263. This results in a ratio of the number of weights to the number of active units of 234.2.

one extreme scenario, one chooses a network topology that provides one large hidden layer and no shortcut connections, in the other extreme scenario one chooses many small hidden layers which are fully connected with each other. The pre-experiments revealed that for the reduced version of PAD, the best results (in respect to the training success) were obtained when one chooses several hidden layers nearly the same size as the input layer and provides complete connections between them. Unfortunately, the complete version of PAD/EVA needs so much calculating time for training that only restricted possibilities could be used to vary the topology of EVA's sub-networks. Nevertheless, because of the structural similarity between the reduced and the complete version of PAD it is plausible to assume that the main results for the former one can be transferred to the latter one.

The input layer of EVA-a comprises the complete vector of 108 input units as presented in table 4.4. The output layer consists of eleven units corresponding to the eleven available actions. The first and fourth hidden layer of EVA-a have 48 units, the second and third have 120 units each. This topology was chosen on the basis of experience with forerunners of EVA-a. However, due to the huge demands on calculating capacity in network training it was not possible to carry out systematic experiments, which included the topology of EVA's sub-networks into the process of model fitting. As the experience shows, the exact topology only has a slight influence on the results of the simulation runs described later on, when one keeps the number of units and weights nearly constant. In this respect, the ratio of weights to active units seems to be of more importance. The units of the hidden layers and the output layer are designated as "active units", because they actually transform their input to an activation value, whereas the input units receive their activation value from an external source. The ratio of weights to active units is an indicator for "connectivity" and seems to be a predictor for training success in the domain of EVA/PAD. However, since this claim is only based on a few observations, I won't go deeper into it.

To sum up, EVA-a has 347 active units and 81263 weights[19], and therefore, a "connectivity ratio" of 234.2. The complete number of units amounts to 455.

4.4.2 EVA-b

As fig. 4.3 shows, EVA-b has only one output unit. This output unit corresponds to the *Delete* operator. Because the external situation-action memory does not comprise the *Delete* operator and is not intended for influencing its evaluator, the external memory is omitted in the input vector for EVA-b. Therefore EVA-b has only 97 input units, in accordance with table 4.4.

Like EVA-a, EVA-b is fully connected too. It has three hidden layers – the first consists of 36 units, the second of 97, and the third again of 36 units. Further, EVA-b has two context layers for the third hidden layer and the output layer. Because of this topological characteristic, EVA-b belongs to the class of hierarchical Elman networks (Zell, 1997). Hierarchical Elman

[19] For EVA-a, EVA-b, and EVA-c the threshold θ_j is replaced by an additional "on-neuron" and an input weight corresponding to this neuron. Therefore the threshold values θ_j are counted as weights.

networks have recurrent connections which are implemented in the form of context layers. Context layers can be assigned to every hidden layer and to the output layer. They are always of the same size as the layer to which they are assigned (their basis layer). They work as a supplementary input: Each unit of the context layer sends its activation to every unit of the basis layer. The activation of each unit in the context layer, on the other hand, is determined by the activation of the corresponding unit in the basis layer in the preceding processing cycle.

Such recurrent connections extend the capabilities of multi-layer networks, so that they are able to recognize chronological information in series of input patterns. Elman (1990) gives several examples even from a psychological point of view.

The following general settings were designed for EVA-b, as well as for EVA-c: In the first processing cycle, when the simulation run or training for a PAD task starts, the activation values of the context layers are set to 0.0. The parameter λ, which influences the storage characteristics of context layers, amounts to 0.0 for EVA-b and EVA-c, so that the context layers are handled in the way described above (which is the simplest way with the least assumptions).

As pointed out for EVA-a, the precise topology of EVA-b was determined by experience and technical considerations. Altogether, EVA-b has 170 active units and 26406 weights, and therefore, a "connectivity ratio" of 155.3. The complete number of units (active units plus context and input units) amounts to 304.

Independent of this technical specification the question arises, why EVA-b was conceptualized as recurrent network. As we will see in later sections, human subjects like to concatenate *Delete* operators. They build "delete chains" of an average size of two *Delete* operators (see

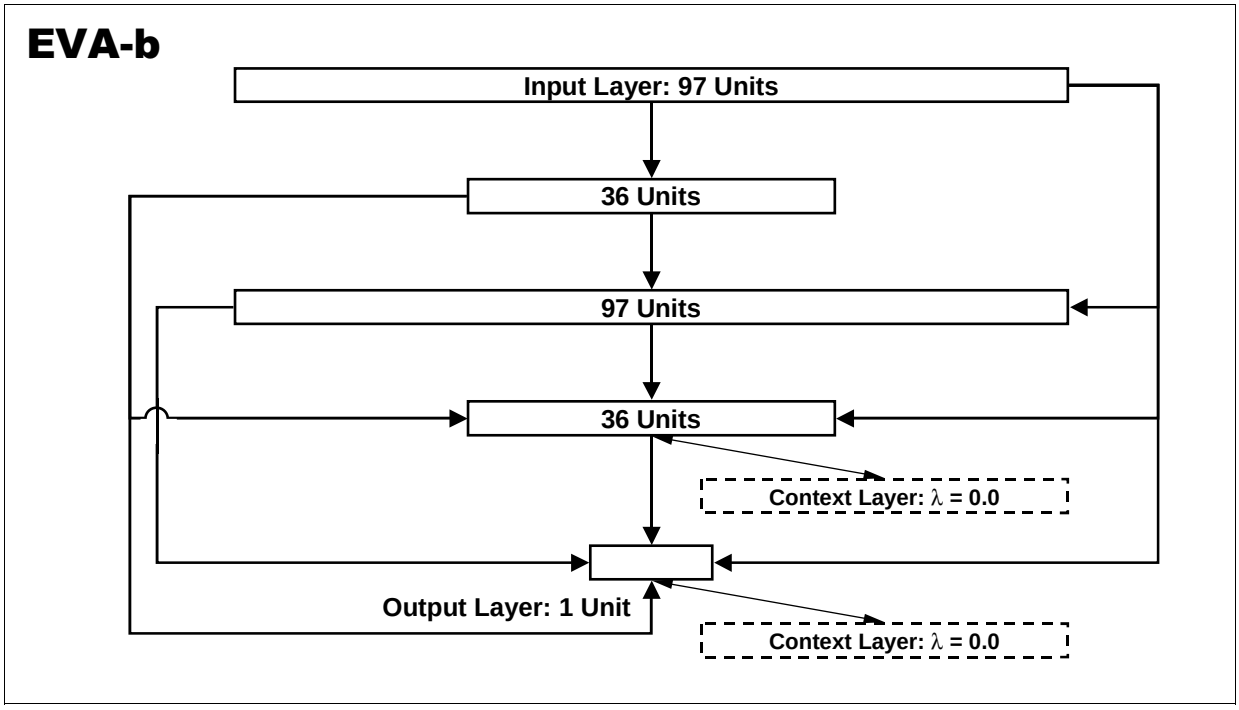

Fig. 4.3: Structure of EVA-b. The unit-to-unit connections are replaced by arrows indicating the connections between entire layers. EVA-b is a fully connected hierarchical Elman network with two context layers. It has 170 active units in its hidden layers and output layer. The number of weights amounts to 26406. This results in a ratio of the number of weights to the number of active units of 155.3.

section 6.1.2). During the development of EVA, it became clear that such behavior cannot be simulated by pure feed-forward networks. Apparently, delete chains are triggered by a specific situation, and afterwards they are maintained independently of the changing situation by inner determinants. In the field of multi-layer networks such inner determinants can be modeled by recurrent connections which inform the network about its prior inner state. Thus, to be able to simulate delete chains EVA-b was equipped with recurrent connections. In respect to this general topological characteristic, EVA-b was designed according to results obtained in the first stage of model fitting (see sections 5.3.1 and 6.1).

4.4.3 EVA-c

The structure of EVA-c is shown in fig. 4.4. The one and only output unit of EVA-c is assigned to the evaluator for the *Finish* operator. According to table 4.4, EVA-c has only 36 input units. The input vector for EVA-c comprises all necessary information to determine the rating of the current plan. As the main finish criterion is the fulfillment of as many appointments as possible (thus, to finish with one of the optimal plans), the current rating is the best indicator for that purpose. If all appointments scheduled in a PAD task can be carried out in the optimal plan(s), then the rating gives unequivocal information as an indicator. For all other PAD tasks, this indicator is not as clear, but quite sufficient, if one takes the general blurredness of neural

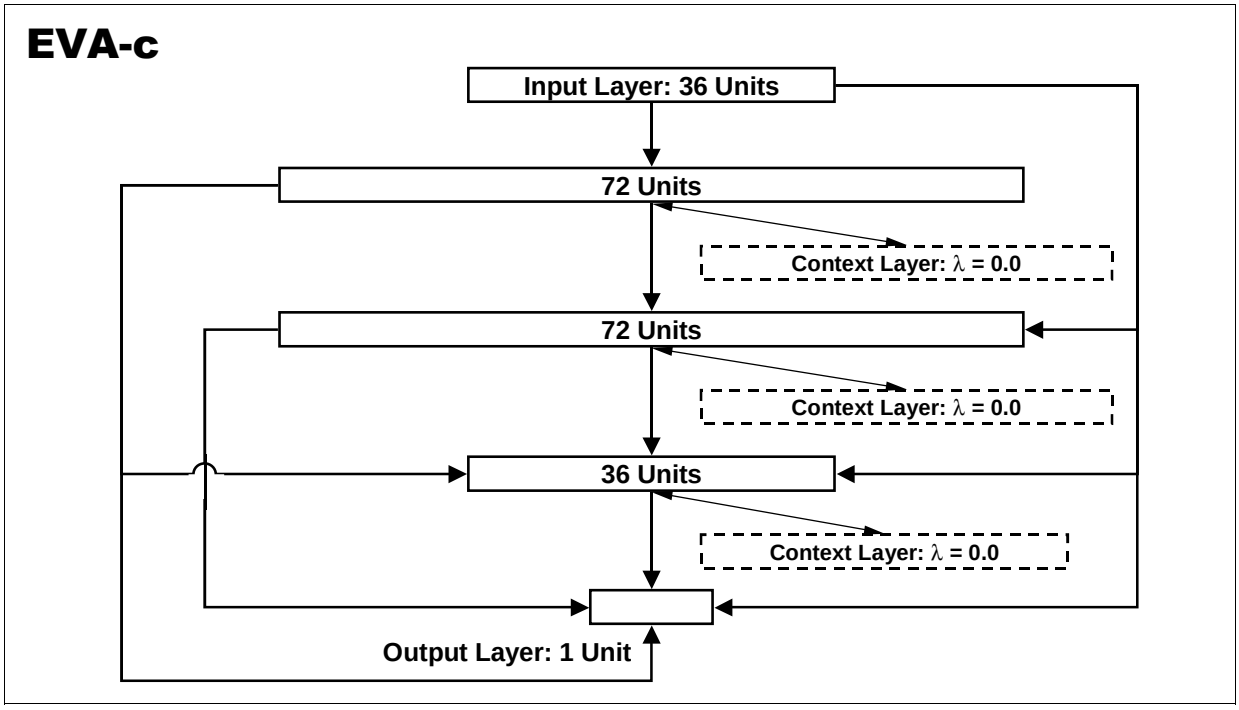

Fig. 4.4: Structure of EVA-c. The unit-to-unit connections are replaced by arrows indicating the connections between entire layers. EVA-c is a fully connected hierarchical Elman network with three context layers. It has 181 active units in its hidden layers and output layer. The number of weights amounts to 28909. This results in a ratio of the number of weights to the number of active units of 159.7.

networks into account. For these reasons as well as for economical reasons the input for EVA-c was reduced to 36 input units.

EVA-c itself has three hidden layers. The first two layers have 72 units each, the third 36 units. EVA-c is completely connected like EVA-a and EVA-b. In addition, all hidden layers are each provided with a context layer, so that EVA-c is a hierarchical Elman network like EVA-b.

As stated for the other sub-networks, the precise topology of EVA-c was also determined by experience and technical considerations. Altogether, EVA-c has 181 active units and 28909 weights, and therefore, a "connectivity ratio" of 159.7. The complete number of units (active units plus context units plus input units) amounts to 397.

EVA-c was designed as recurrent network to model motivational influences on the "Finish operator". These influences were observed in the data obtained by human subjects. Thus, also results from model fitting were used for network design for EVA-c. As the data show, many subjects finish the planning process without having reached the optimal plan. This hasty finishing may have several reasons. First, the interest or desire to work on the PAD task has dropped under an individual threshold. Second, the chances of greater success are estimated as quite small after several unsuccessful attempts of further improvement. Third, the pressure, due to the limited time for working on the PAD task has led to finishing. All these possible reasons have one thing in common: They are related to time passing by. Thus, in addition to the pure rational evaluation of the *Finish* operator, EVA-c has to take this time component into consideration. In order to be able to fulfill such a demand, EVA-c needs recurrent connections.

4.4.4 EVA-a, EVA-b, and EVA-c Considered Together

Even if it may seem very arbitrary at first glance, which precise topology is chosen for each of EVA's sub-networks, there is one common principle in the number of units designated to hidden layers. The first and last hidden layer have an identical and comparatively small number of units, while the middle hidden layer(s) have at least as many units as the input layer. Because EVA-c has such a small input layer in comparison, this basic scheme was varied, and the input layer was identified with the first hidden layer.

After several attempts with other topologies (e.g., with hidden layers of decreasing size from input to output), this basic scheme was chosen, because it is supposed to have a slight advantage in training success.

In regard to the number of weights and units, EVA-a, EVA-b, and EVA-c are different. These differences are based on the different number of output units and the different (assumed) complexity of the evaluation task(s) demanded by each network.

4.5 Network Training

4.5.1 General Considerations

Section 2.1.1.3 gives a brief introduction to connectionist learning. The foundation for learning in multi-layer networks is the generalized delta rule, on which the backpropagation procedure is based. By using backpropagation, multi-layer networks can be trained to perform a set of input-output-transformations. Before training, such networks are considered “stupid”. The output patterns, which they produce as transformation of certain input patterns, are a result of pure chance. This corresponds to the fact, that network weights are randomly set at the beginning. Only through training do these weights receive values that allow meaningful input-output-assignments. In general, this training takes place by the repeated presentation of input-output-pattern-pairs to the network and corresponding weight adjustments.[20]

The first objective of training is that afterwards the network is capable of handling the training patterns in the desired way. However, for many problems the set of patterns is very large, and therefore, training can only take place with a subset of these patterns. For this reason, it is also desirable that the network is able to produce the correct output pattern for input patterns to which it never has been applied during training. This capability is called “generalization”. To control the generalization abilities of a network during training, the performance of the network is often not only supervised according to the training patterns, but also according to a set of so-called test patterns. In general, the generalization abilities are better the more training patterns are used. When the amount of training patterns is too small, a network will learn these patterns as special cases, but will not derive a general hypothesis about the relationship between input and output patterns.

On the other hand, when the number of training patterns increases, the time needed for one epoch also grows (linearly). An epoch comprises the complete presentation of all training patterns to the network including the according weight adjustments. Often, many epochs are needed to fulfill a certain criterion defined for finishing training. Such criteria mostly rely on the error either for the training set or the test set or both. I will go deeper into this subject in section 4.5.3. Roughly speaking, the number of epochs needed may vary in the range from 10 to 100 000, depending on the learning procedure[21], the network size and topology, the complexity of the problem, the representation chosen for input and output, and the number and selection of training patterns. The training of neural networks is a very demanding job for today’s (desktop) computers[22], when the complexity of the stated problems reaches the level of EVA/PAD.

While section 4.5.3 addresses the questions of the appropriate training procedure for EVA’s sub-networks and section 4.5.4 presents the actual course of training, in section 4.5.2, as beginning, the creation of the training data is described.

---

[20] The following descriptions refer only to multi-layer networks of the "backpropagation-class" and not to connectionist networks in general.

[21] Meanwhile there are many competitors to backpropagation.

[22] Of course, the use of more advanced hardware like SIMD parallel computing systems or specialized neuro processors, allows research on much larger connectionist networks with much larger training sets (see Zell, 1997).

4.5.2 Training Data

4.5.2.1 Basic Principles of Data Generation

A separate set of training and test patterns was generated for each of EVA's sub-networks. Nevertheless, the basic principles of generation are the same for all sub-networks. First, a PAD task is randomly created, that fits several parameter settings, which I will describe later on. Afterwards, on the basis of this PAD task, a sequence of operators is generated, also randomly, but under certain constraints. This sequence is used to determine a corresponding chain of states of the PAD world (by applying these operators one after another). For each state an input vector is constructed, which meets the specifications of table 4.4, that means: It comprises the PAD task, the current state, the preceding operator and reaction, as well as the continuously updated situation-action memory. According to this input vector and to the principles outlined in section 4.3.2, the desired output vector is determined as well. By this way, the input-output-pattern-pairs for training and testing are generated. The procedures needed for this purpose and also for the later presented simulation runs are implemented in a PAD simulation system, that is based on a C++ function library.

Table 4.5: Parameters for the random generation of PAD tasks, their default values, and their meaning

Parameter	Default value	Meaning
General parameters		
MinSchedulApps	5	Minimum number of scheduled appointments in a PAD task
MaxSchedulApps	9	Maximum number of scheduled appointments in a PAD task
StartLocation	3 (Office)	Location in the PAD world, where the subject starts
DayBegin	10:00 a.m.	Time at which the day in the PAD world begins
LastAppBegin	3.00 p.m.	Latest time at which a period of appointment fulfillment can begin
LastAppEnd	6.30 p.m.	Latest time at which a period of appointment fulfillment can end
MaxTaskDuration	90 min.	Maximum period the fulfillment of an appointment can take
ConfMoment	true	Flag, if the earliest and latest time of task starting are identical for the conference room
pBTequDB	0.5	Probability for each appointment, that its period of fulfillment begins at DayBegin
MeanTimeRange	222 min.	Average length of the period of appointment fulfillment
Parameters regarding the optimal plan(s) for each generated PAD task		
MinNoOptPlans	1	Minimum number of optimal plans
MaxNoOptPlans	1	Maximum number of optimal plans
MinInOPLoc	3	Minimum number of locations visited in the optimal plan(s)
MaxInOPLoc	9	Maximum number of locations visited in the optimal plan(s)
MinOPRating	3	Minimum rating of the optimal plan(s)
MaxOPRating	72	Maximum rating of the optimal plan(s)
OPIncludeCar	true	Flag, if the optimal plan(s) include(s) the *Drive-by-car* action

Creation of PAD tasks

The random generation of PAD tasks is controlled by several parameters, which are presented in table 4.5. These parameters are chosen in a way so that they meet the basic characteristics of the predefined PAD tasks 1 to 16 in the "real" PAD system. The default values presented in table 4.5 are subject to slight variations for each of EVA's sub-networks. These changes will be addressed in the corresponding sections.

Creation of sequences of operators

The generation of these sequences is based on a random choice of operators (always finished by the *Finish* operator). This random choice can be influenced by several parameters, so that the generated sequences can differ widely in their quality. In one extreme, such a sequence is nothing more than a totally random succession, in the other extreme, the sequence leads to one of the optimal plans in the shortest possible way. Usually, a medium way was chosen in generation of the training data. Since the precise parameter settings differ widely between EVA-a/EVA-b on one side, and EVA-c on the other side, the following table 4.6 only shows the parameters and their meanings; the parameter values are given later on.

The parameters MaxPDelete, MaxPCar, MaxPInvalid and ExpPOpt need some explanations in advance. Before any sequence is created, four probabilities are fixed in advance: The probability for any operator in this sequence to be a Delete operator, a Drive-by-car operator, an invalid operator (according to table 4.3), or an operator leading to an optimal plan. The first three probabilities are determined randomly, and in doing so, MaxPDelete, MaxPCar, and MaxPInvalid determine the corresponding maximum probabilities. Afterwards, the probability for an operator to be an optimal one (pOpt) is determined. (Of course, it is always taken into consideration that the sum of probabilities has to be equal to 1.0.) The residual probability is designated to the case in which the operator is chosen completely at random. By ExpPOpt pOpt may be increased to the debit of this residual probability.

Table 4.6: Parameters for the random generation of operator sequences, and their meaning

Parameter	Meaning
MaxiSequence	Maximum number of operators in a sequence; after MaxiSequence operators the sequence is compulsorily finished by a *Finish* operator. Also, a sequence is always finished when an optimal plan has been reached.
MaxPDelete	Maximum Probability which can be determined for the *Delete* operator
MaxPCar	Maximum Probability which can be determined for the *Drive-by-car* operator
MaxPInvalid	Maximum Probability which can be determined for the application of an invalid operator
ExpPOpt	A value larger than 1.0 decreases the probability for the application of operators leading to an optimal plan, a value smaller than 1.0 increases this probability.
pRepeatDelete	Probability that a *Delete* operator is followed by another *Delete* operator (in addition to the basic rules of sequence generation)
OnlyInvalid	Flag, if only invalid *Delete* operators are used as operators

4.5.2.2 Training Data for EVA-a/EVA-b

The training data produced for EVA-a and EVA-b is based on the same randomly generated PAD tasks and the same random sequences of operators. The difference between EVA-a and EVA-b is the different format of their input vector and that they are assigned to different output units.

The parameters for PAD task generation take the default values of table 4.5. Furthermore, there are two additional features. First, in every seventh PAD task OPIncludeCar is set to false. Second, in every ninth PAD task the priorities of all appointments are set to equal levels. Through these variations, the scope of learning experience for the networks is widened.

The parameters for the generation of operator sequences take the values shown in table 4.7. As this table reveals, there are two different configurations. The left configuration produces planning processes with many sub-optimal or even invalid operator applications, which facilitate the training of the *Delete* evaluator and also of the influence of the preceding operator and reaction and of the external situation-action memory. On the other hand, the right configuration leads (nearly) always directly to the optimal plan. In such a case the sequences are comparatively short, and many different PAD tasks can be presented to the networks during training without breaking up the size of the set of training patterns.

EVA-a

The output provided for the training of EVA-a is based on the evaluators described in section 4.3.2. However, in addition, in order to train the influence of the external situation-action memory, minor modifications are necessary. Namely, as the output units of EVA-a correspond to the eleven available actions, and the <MemoryUnits> are also assigned to one action each, every output value in the training data is multiplied with the difference between 1.0 and the value of the corresponding <MemoryUnit>. Through this procedure, EVA-a is trained to process the external memory as an inhibitor on the application of actions. Whenever EVA-a enters a situation for the second time, the output unit assigned to the action carried out at the first time will be inhibited. The strength of inhibition can be regulated by the value of MemForgetFactor.

Table 4.7: Parameter values for the random generation of operator sequences for the training data of EVA-a and EVA-b

Parameter	Configuration	
	for 40 % of the generated PAD tasks	for 60 % of the generated PAD tasks
MaxiSequence	40	40
MaxPDelete	0.15	0.0
MaxPCar	0.15	0.0
MaxPInvalid	0.3	0.0
ExpPOpt	0.7	0.001
pRepeatDelete	0.75	0.0
OnlyInvalid	false	false

This way, EVA is prevented from “running in circles”, as was the case for the first versions of EVA, which did not have external memory. Then, often one specific action and the Delete operator alternated in the sequence of operators produced by EVA without ever finishing. This did not correspond at all to the behavior observed in human subjects.

Finally, the training set for EVA-a comprises 150297 input-output-pattern-pairs, which are based on 8450 different PAD tasks. The number of test patterns amounts to 6314 (equivalent to 350 PAD tasks). Thus, the average length of each operator sequence (for one PAD task each) amounts to 17.8.

EVA-b

According to the remarks in section 4.4.2, EVA-b is supposed to produce delete chains. Apart from the recurrent architecture of EVA-b, this capability has to be supported by the structure of the training data. Therefore, the *Delete* evaluator provided for training is further modified in comparison to the output specified in section 4.3.2. Namely, whenever the *Delete* operator occurred in the randomly generated sequence of operators in the previous cycle (whereby, in the current cycle, the preceding operator in the input is the *Delete* operator), the *Delete* evaluator is modified in the following way:

$$Ev_{Delete,\,t} = (1.0 - \mathrm{DelRepeatFactor}) \cdot Ev_{Delete,\,t} + \mathrm{DelRepeatFactor} \cdot Ev_{Delete,\,t-1} \qquad (4.1)$$

$Ev_{Delete,\,t}$ is the value of the Delete evaluator in the current cycle, $Ev_{Delete,\,t-1}$ is its value in the preceding cycle.

In this formula, the parameter DelRepeatFactor (range within [0.0; 1.0]) is introduced. This parameter determines, how strong a preceding *Delete* operator influences the current *Delete* evaluator. A value of 0.0 for DelRepeatFactor leads to no modifications at all, a value of 1.0 results in the current *Delete* evaluator being equal to the preceding *Delete* evaluator. In the latter case, EVA-b would be trained to rely totally on its preceding evaluation of the Delete operator whenever this evaluator has reached the maximum activation of all output units in the preceding cycle. Actually, the output patterns in the training data for EVA-b were generated with a DelRepeatFactor of 0.75. This value was chosen due to results of the first stage of model fitting (see section 5.3.1).

The training set for EVA-b comprises 88325 input-output-pattern-pairs which are based on 5000 different PAD tasks. The number of test patterns amounts to 9084 (equivalent to 500 PAD tasks).

4.5.2.3 Training Data for EVA-c

For the training data for EVA-c, the parameters for PAD task generation also receive the default values of table 4.5. In generating the sequences of operators, three different configurations were mixed in the training data. The parameter settings for these configurations are shown in table 4.8. Every configuration was used for one third of the generated PAD tasks.

Table 4.8: Parameter values for the random generation of operator sequences for the training data of EVA-c

Parameter	Configuration		
	(a)	(b)	(c)
MaxiSequence	120	120	120
MaxPDelete	0.15	0.0	–
MaxPCar	0.15	0.0	–
MaxPInvalid	0.3	0.0	–
ExpPOpt	1.0	0.001	–
pRepeatDelete	0.75	0.0	–
OnlyInvalid	false	false	true

Configuration (b) leads straightforward to optimal plans by which EVA-c can learn, when the optimal plan is reached and, therefore, a maximum value for the *Finish* evaluator is demanded. By configuration (a), sequences are generated that show an alternation between decreasing and increasing distance to the optimal plan and, therefore, help to train alternating evaluations of the *Finish* operator. In contrast, configuration (c) only causes sequences of *Delete* operators, which have no effect on the state of the PAD world since they are invalid (at the beginning of the planning process there is nothing to delete). In this case, the influence of time on the evaluation of the Finish operator (see section 4.4.3) can be specifically trained, as pointed out in the following.

In addition to the standard evaluation of the *Finish* operator as described in section 4.3.2, the output unit of EVA-c also reflects the influence of time on the *Finish* evaluator. In this regard, time means not the time within the PAD world, but the "real" time passing by for the subject in front of the computer screen. This kind of time is modeled by the transition of one processing cycle to the next. As shown in section 4.3.2, with ongoing time the readiness for finishing grows, and thus, this observation taken from empirical observations has to be reflected by a *Finish* evaluator, which increases from processing cycle to processing cycle.

Two parameters serve to determine this influence of time on the *Finish* evaluator. First, EndThreshold determines the number of cycles after which the value of the *Finish* evaluator always amounts to 1.0. Second, ExpET (range: [0.0; ∞[ ) regulates how strongly the *Finish* evaluator is increased in the cycles before (in comparison to its "normal" value). The larger the value of ExpET, the less the evaluation of the *Finish* operator is changed until a few cycles before EndThreshold. The smaller the value of ExpET, the earlier and the stronger the *Finish* evaluator is increased in course of the ongoing processing cycles. For the actual training data for EVA-c the following values were chosen: ExpET = 5.5 (which is comparatively large), EndThreshold = 119. These values are based on results from the first stage of model fitting (see section 5.3.1).

Thus, configuration (c) helps EVA-c to learn the pure influence of time on the desired value of the *Finish* evaluator. As pointed out in section 4.4.3, this learning can only take place because of the recurrent connections of EVA-c.

Obviously, the characteristics of the training data needed for EVA-a/EVA-b on the one hand and EVA-c on the other hand are quite different. This is the main technical reason, why a separate sub-network was specified for the *Finish* evaluator. Theoretically, one could be able to train all output units of EVA together in one joined network, but practically, the calculating demands for such an endeavor would exceed the limits of the computer equipment available for this study.

Finally, the training set for EVA-c comprises 86812 input-output-pattern-pairs which are based on 1200 different PAD tasks. The number of test patterns amounts to 13604 (equivalent to 200 PAD tasks). Thus, the average length of each operator sequence (for one PAD task each) amounts to 71.7.

4.5.3 Training Algorithm

For the training of EVA's sub-networks and their use within the PAD simulation system a C++ class library was developed. An executable for the Win32 console ("NWRun"), which is based on this library, was used for network training. The user guide for NWRun is provided in appendix D. In addition, both NWRun and the complete source code of the library are available for download at "www.wolframschenck.de/NetworkApplication.htm".

The library provides five different training algorithms for multi-layer networks with numerous possible parameter settings: Standard backpropagation (BackProp), backpropagation with momentum term (BackPropMom), quickpropagation (QuickProp), resilient propagation (RProp), and backpercolation (Perc) (algorithms taken from Zell, 1997; for RProp see also Riedmiller & Braun, 1993) are available. The choice of the algorithm taken for EVA is discussed in the following sections.

4.5.3.1 Choosing the Best Training Algorithm

The main goal of training is to reduce the difference between the output actually produced by a network (o) and the desired output (t). Therefore, based on the difference ($o_{j,p} - t_{j,p}$), for each pattern pair p and each output unit j a global error E is calculated as follows:

$$E = \frac{1}{2 \cdot N_j \cdot N_p} \sum_j (o_{p,j} - t_{p,j})^2 \qquad (4.2)$$

E is the global error the network produces, N_j is the number of output units, N_P is the number of patterns, $o_{p,j}$ is the actual activation of output unit j for pattern p, $t_{p,j}$ is the desired activation of output unit j for pattern p.

The size of E is a good indicator for the quality of the network's output in respect to the desired output. As pointed out in section 2.1.1.3, training procedures like backpropagation try to

minimize this error by adjusting the network weights according to the gradient of E. Ideally, by such a procedure the error decreases from epoch to epoch and finally reaches a value near zero. (For comparison: EVA's sub-networks, when their weights are randomly set, produce errors in the range from 0.075 to 0.125.)

The success of training can be assessed by several indicators. First, the error value, the network finally converges to, should be as small as possible. In this respect, it is fatal if the training keeps stuck in a local minimum of the error function E. Second, the number of epochs needed to fall short of a certain error level should be as small as possible too. (Since different training algorithms need different calculating time for each epoch, an even more objective standard of comparison is the calculating time needed for falling short of a certain error level.) Third, the test error should be as close to the training error as possible. This refers to the generalization capabilities of the trained network. The test error is obtained on the set of test patterns (see section 4.5.1), the training error on the set of training patterns.

By using these three indicators (training error, number of needed epochs, test error) different network topologies and different training algorithms can be compared. As results of pre-experiments with a reduced version of PAD (called "Mini-Plan-A-Day" or MPAD) showed, RProp is superior to all other training algorithms from the tested selection (which comprises BackProp, BackPropMom, QuickProp, RProp, and Perc). For example, a direct comparison between RProp and BackPropMom was carried out for a two layer network with two shortcut connections. While the training error was halved after 1500 training epochs by BackPropMom, the same error reduction could be achieved by RProp in about 70 epochs. BackProp, the pure standard algorithm for backpropagation networks (see section 2.1.1.3), shows an even poorer performance. In contrast, for MPAD, QuickProp is nearly as good as RProp. Perc shows favorable results in the first epochs of training, but the algorithm remains stuck in a local minimum very quickly. In addition, it is very difficult to determine the optimum values for the adjustable parameters of Perc.

However, not only with regard to the final training error and the number of needed epochs, but also with regard to its generalization capabilities, RProp is superior to other algorithms. Furthermore, RProp is very robust concerning its parameter settings. For most problems, the recommended settings (Riedmiller & Braun, 1993) yield optimum convergence time.

Since all training algorithms of the backpropagation family are not very plausible from a biological or psychological point of view (see section 2.1.1.3), RProp was chosen only due to technical considerations. The psychological interest on EVA is, therefore, restricted to the networks resulting from training, but cannot be extended to the course of training itself.

4.5.3.2 Description of RProp

Algorithm

In standard backpropagation, the weight changes Δw_{ij} are determined by the product of a fixed learning constant η (the learning-rate) and the partial derivative $\frac{\delta E}{\delta w_{ij}}$. If η is too small, weight

adaptation takes place very slowly, if η is too large, weights begin to oscillate between different values. To overcome this dependence on the global learning-rate, some improved variants of backpropagation work with a local adaptation of η, so that every weight has its own learning-rate η_{ij}, which is changed according to the observed behavior of the error function. Examples for such algorithms are the Delta-Bar-Delta technique (Jacobs, 1988; cited in Zell, 1997) or the SuperSAB algorithm (Tollenaere, 1990; cited in Zell, 1997).

Even if these algorithms are superior to standard backpropagation, they ignore the fact that a sudden change of the partial derivative $\frac{\delta E}{\delta w_{ij}}$ can push the weight into the wrong direction, so that the laborious adaptation of the specific learning rate is in vain. RProp was developed to overcome this weakness (Riedmiller & Braun, 1993). Riedmiller and Braun introduced an individual update value Δ_{ij} for each weight w_{ij}. This update value is changed due to the partial derivative $\frac{\delta E}{\delta w_{ij}}$, but does not depend directly upon it. The following formula gives the calculating rule for Δ_{ij} after each epoch (in RProp the update values and weights are changed every time the whole pattern set has been presented once to the network):

$$\Delta_{ij}^{t} = \begin{cases} \eta^{+} \cdot \Delta_{ij}^{t-1} & , \text{ if } \frac{\delta E}{\delta w_{ij}}^{t-1} \cdot \frac{\delta E}{\delta w_{ij}}^{t} > 0 \\ \eta^{-} \cdot \Delta_{ij}^{t-1} & , \text{ if } \frac{\delta E}{\delta w_{ij}}^{t-1} \cdot \frac{\delta E}{\delta w_{ij}}^{t} < 0 \\ \Delta_{ij}^{t-1} & , \text{ else} \end{cases} \qquad (4.3)$$

$$\text{where } 0 < \eta^{-} < 1 < \eta^{+}$$

The superscript t indicates the current epoch, while the superscript t-1 indicates the preceding epoch. η^{-} and η^{+} are the decrease and increase factor, respectively.

To verbalize this adaptation rule, I would like to cite Riedmiller and Braun (1993, p. 587): “Every time the partial derivative of the corresponding weight w_{ij} changes its sign, which indicates that the last update was too big and the algorithm has jumped over a local minimum, the update value Δ_{ij} is decreased by the factor η^{-}. If the derivative retains its sign, the update value Δ_{ij} is slightly increased [by the factor η^{+}] in order to accelerate convergence in shallow regions.”

After determination of the update values, the weight changes Δw_{ij} are calculated and carried out according to the following simple rule (Riedmiller & Braun, 1993, p. 587): “If the derivative is positive (increasing error), the weight is decreased by its update value, if the derivative is negative, the update value is added.”

$$\Delta w_{ij}^{t} = \begin{cases} -\Delta_{ij}^{t} & , \text{ if } \frac{\delta E}{\delta w_{ij}}^{t} > 0 \\ +\Delta_{ij}^{t} & , \text{ if } \frac{\delta E}{\delta w_{ij}}^{t} < 0 \\ 0 & , \text{ else} \end{cases} \qquad (4.4)$$

$$w_{ij}^{t+1} = w_{ij}^{t} + \Delta w_{ij}^{t} \qquad (4.5)$$

However, there is one exception, when a “backtracking” weight-step should be carried out:

$$\Delta w_{ij}^{t} = -\Delta w_{ij}^{t-1} \text{ , if } \frac{\delta E}{\delta w_{ij}}^{t-1} \cdot \frac{\delta E}{\delta w_{ij}}^{t} < 0 \qquad (4.6)$$

When the sign of the derivative changes, the most likely explanation is that the previous step was too large and the minimum was missed. For that reason the weight change is undone by this backtracking weight-step. Furthermore, in this case one should set $\frac{\delta E}{\delta w_{ij}}^{t-1} := 0$ for the next adaptation of the update values to avoid a double punishment of Δ_{ij}^t.

In contrast to other variants of backpropagation, only the sign of the partial derivative is used to perform both learning and adaptation. Riedmiller and Braun (1993, p. 588) write, that "this leads to a transparent and yet powerful adaptation process, that can be straight forward and very efficiently computed with respect to both time and storage consumption."

Parameters

For RProp, there are five adjustable parameters: The factors η^- and η^+, the initial value Δ_0 for the update values, and the lower and upper limit of the range to which the update values are restricted (from Δ_{min} to Δ_{max}). Riedmiller and Braun (1993) recommend the following standard values: $\eta^- = 0.5$, $\eta^+ = 1.2$, $\Delta_0 = 0.1$, $\Delta_{min} = 1.0 \cdot 10^{-6}$, $\Delta_{max} = 50.0$. According to their experiments with RProp, these values are the optimum ones for most problems. They found different optimum values for different problems only for Δ_0. However, the range for Δ_0 within which RProp works almost optimally is so broad that one need not worry about this parameter.

For the training of EVA's sub-networks, the standard parameter settings were adopted. Only Δ_0 was set to 0.04. The weights were initialized before training to random values in the range [-0.2; 0.2]. As experiments with MPAD revealed, this initialization range, together with a value of 0.04 for Δ_0, makes a good starting point for training.

4.5.3.3 Weight Decay

For the training of EVA's sub-networks the standard RProp algorithm was supplemented by another procedure, called weight decay. Weight decay may be used together with many variants of backpropagation to enhance the generalization capabilities of the resulting networks (Zell, 1997). By weight decay the size of the weights is diminished in every epoch by a certain factor. Thus, the rule for weight update is modified according to the following formula:

$$w_{ij}^{t+1} = (1 - d) \cdot w_{ij}^t + \Delta w_{ij}^t \tag{4.7}$$

d is the decay value in the range [0.0; 0.03]. Larger values for *d* are not recommendable.

As pre-experiments with MPAD showed, weight decay actually decreases the difference between the training error and the test error. This advantage goes hand in hand with the disadvantage that the training error decreases more slowly. Further, the value to which the training error converges is higher than it is without weight decay. A close inspection of the course of training revealed, that the training stagnates at the point, when the weight changes Δw_{ij} have become too small to overcome the impact of the weight decay $d \cdot w_{ij}$. When one chooses a

smaller value for *d* at this point, the training error decreases again until the next equilibrium is reached.

To benefit from the advantages of weight decay without suffering from its disadvantages, a mixed strategy was used for the training of EVA's sub-networks. The initial value of *d* was decreased during the course of training by an automatic weight decay adaptation. This automatic adaptation procedure was used to monitor the course of training. Whenever the reduction of the training error after a certain number of epochs fell short of a certain threshold, the decay parameter *d* was reduced by multiplication with a decay adaptation factor in the range [0.0; 1.0]. In NWRun, the precise behavior of the automatic adaptation is controllable through five parameters. First, WDA_NumEpochs determines the number of epochs on which the comparison of the training error is based. Second, to avoid dependence on single error values which can be subject to sudden and short-lived changes, the training error at the beginning and at the end of the comparison period is not determined by a single value, but by the median of the error values of WDA_NumMedian epochs. Third, the threshold which cannot be transgressed, is defined by WDA_MinDiff (within the range]-∞; 1.0]). If the current error median is larger than the product of (1 – WDA_MinDiff) with the error median WDA_NumEpochs epochs ago, then an adaptation of the decay parameter *d* is carried out. This adaptation takes place by multiplication of *d* with the fourth parameter, WDA_DecayFactor (within the range [0.0; 1.0]). And last, but not least, there is the initial value of *d*, represented by WDA_DecayInit.

WDA_NumEpochs was set to 200 for all of EVA's sub-networks, WDA_NumMedian was set to 20, and WDA_DecayFactor was set to 0.6. Only WDA_MinDiff and WDA_DecayInit differed between the sub-networks. WDA_MinDiff was set to 0.05 for EVA-b and EVA-c, and for EVA-a it was set to 0.075. WDA_DecayInit was set to 0.005 for EVA-c, to 0.003 for EVA-b, and to 0.0018 for EVA-a. Since EVA-a has the largest training set, which already enhances generalization performance, a larger value for WDA_MinDiff and a smaller value for WDA_DecayInit were chosen than for EVA-b and EVA-c. Through the larger value for WDA_MinDiff, adaptation of the decay parameter *d* was intended to take place earlier. Together with the comparatively small initial value for *d*, this should speed up the decrease of the training error a little. In general, the parameter values were chosen due to the observation of several courses of training with MPAD on the one hand and with forerunners of EVA's sub-networks on the other hand. The main goal was to achieve a well-balanced relation between convergence time and generalization performance.

4.5.4 Course of Training

In the following, the actual course of training for EVA's sub-networks is presented. For each sub-network only one weight configuration was calculated through training due to two reasons: First, training takes a large amount of time (e.g., for EVA-a more than 280 hours on a 700-Mhz-AMD-Athlon-CPU were needed). Second, as the training of the forerunners of EVA's current sub-networks showed, for networks of this size and with such a large training set, the course of

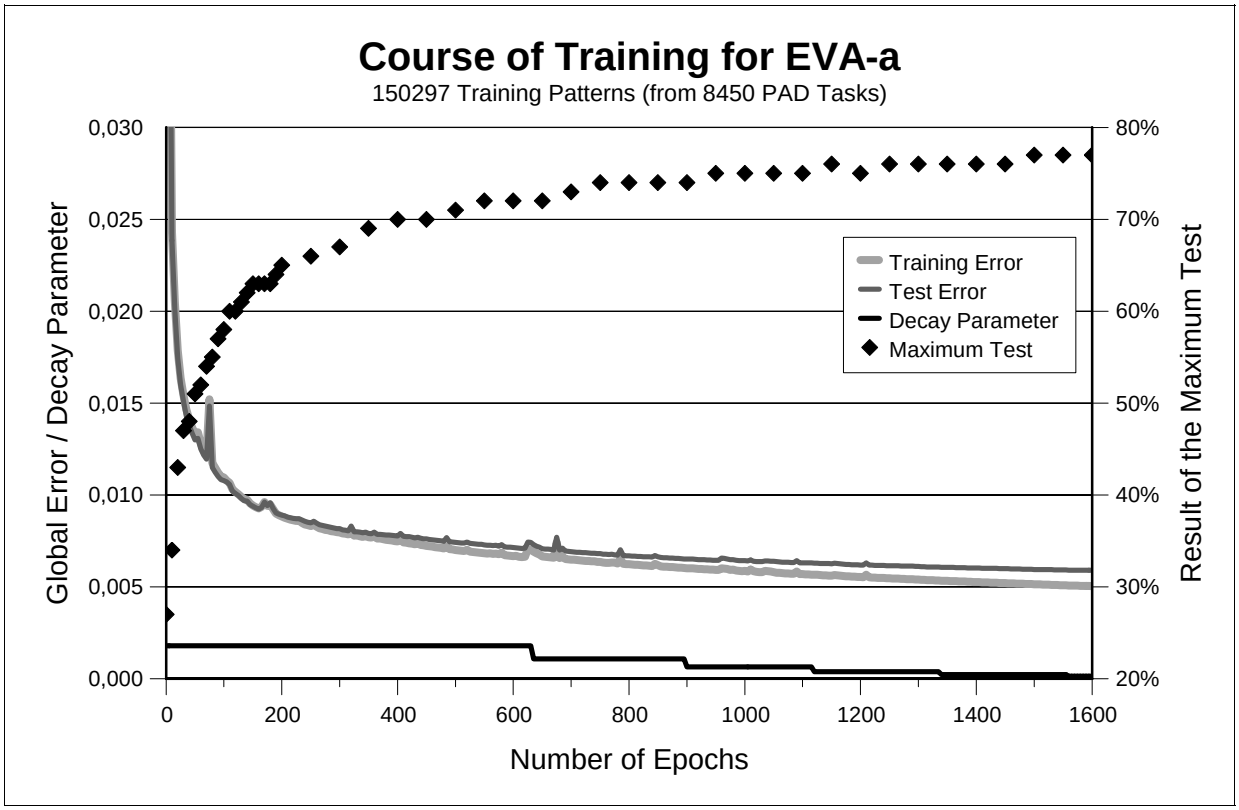

Fig. 4.5: Course of Training for EVA-a. Before training, after random initialization of weights, the training and test error amounted to 0.077.

training differs only slightly between different network instances. Even for most of the performance indicators assessed in model fitting (see section 6.1), only minor differences were observed in pre-experiments.

Figures 4.5 to 4.7 show the course of training for EVA-a, EVA-b, and EVA-c, respectively. Every diagram displays one curve for the training error (light gray) and one curve for the test error (dark gray). Furthermore, the decay parameter *d* is sketched in. For EVA-a, an additional indicator for training success was calculated: The "maximum test" indicator. As the operator that EVA will carry out is determined by the activation maximum of the output units, the maximum test checks for the output units of EVA-a. More precisely, it checks if the output unit with maximum activation of the produced output is identical to the output unit with maximum activation of the desired output. This test is carried out with the patterns of the test set, and the resulting percentage corresponds to the proportion of correct assignments. In figure 4.5, the maximum test indicator is presented for discrete epochs.

EVA-a was trained for 1600 epochs, EVA-b and EVA-c for 2000 epochs. Training was halted when no further success in significantly reducing the test error was expected.

EVA-a

The course of training for EVA-a is presented in figure 4.5. The curves for the training and test error show a logarithmic characteristic. While the training error is reduced from 0.077 to 0.0077 in the first 350 epochs, in the following 1250 epochs only a reduction to 0.005 could be

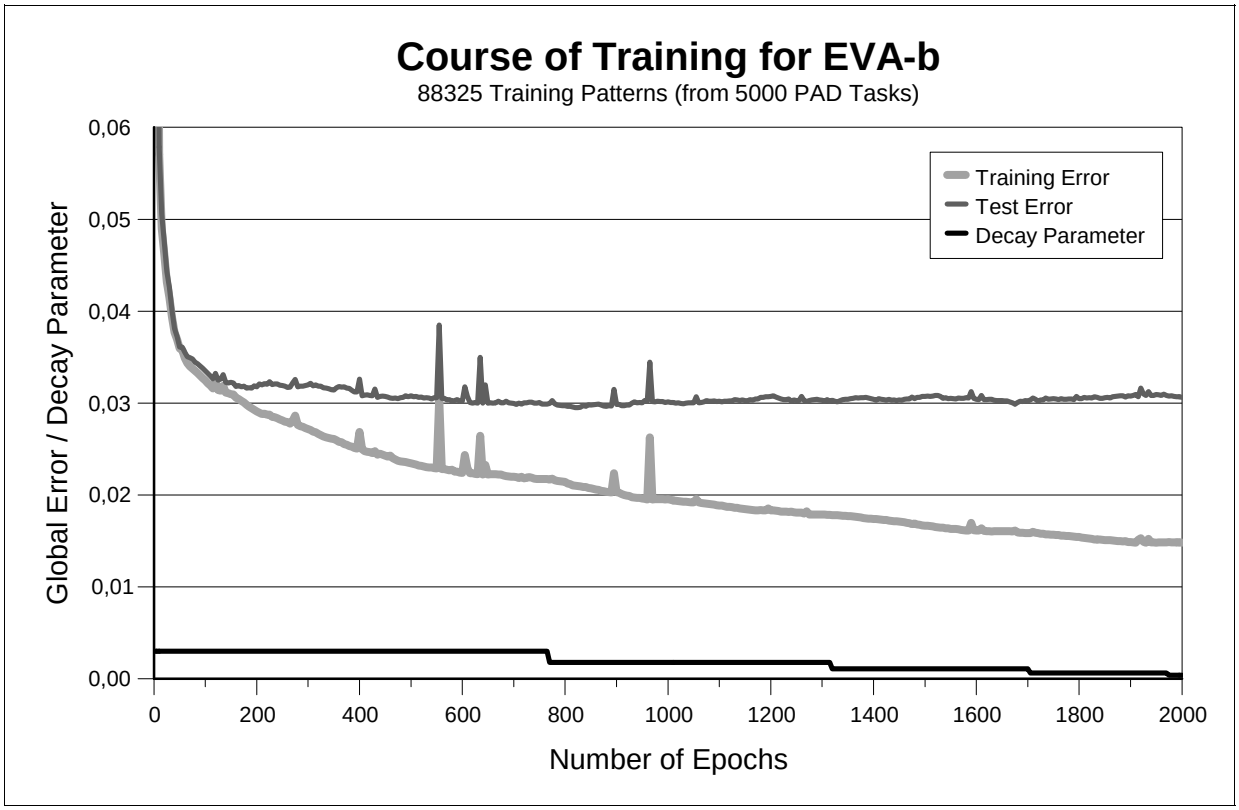

Fig. 4.6: Course of Training for EVA-b. Before training, after random initialization of weights, the training and test error amounted to 0.11.

achieved. It is clearly visible, how the network converges to a certain performance level in the process of training. The generalization capabilities are satisfactory. In epoch 1600, the training error amounts to 0.0050, while the test error amounts to 0.0059. The ratio of the number of free network weights (81263) to the number of training patterns (150297) is about 0.5. Pre-experiments indicated, that such a ratio represents a good compromise between convergence time (which decreases with an increasing number of training patterns) and generalization performance (which increases with an increasing number of training patterns).

In the maximum test, the resulting curve has also a logarithmic characteristic, but with the opposite sign in comparison to the global error curves. The best result for the maximum test is reached in epoch 1540, namely 77.4%. For comparison: A forerunner of EVA-a with 64271 weights and 118651 training patterns reached an optimum value of 76.6% in epoch 1720. Training of this forerunner was one and a half times faster than for EVA-a. Thus, by time factor 1.5 an improvement of only 0.8% in the maximum test could be achieved. Obviously, further enlargement of EVA-a and of the training set would only result in minimal improvements in the performance of the resulting network. The convergence zone of the PAD problem is reached for the given representation of input and output (at least for EVA-a).

EVA-b / EVA-c

The course of training for EVA-b and EVA-c is shown in figure 4.6 and 4.7, respectively. Both networks show a training characteristic different to EVA-a. Quite early, after about 150 epochs,

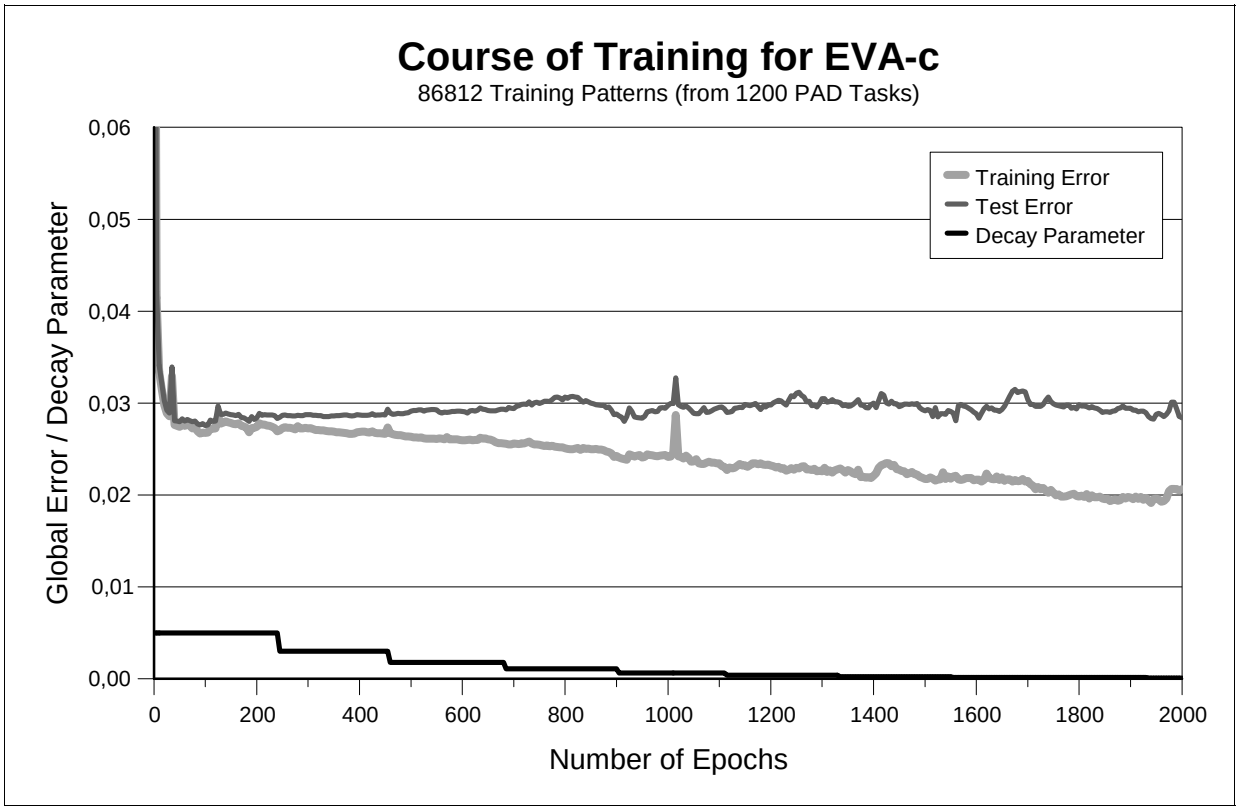

Fig. 4.7: Course of Training for EVA-c. Before training, after random initialization of weights, the training and test error amounted to 0.13.

the test error reaches a level from which it does not fall significantly in the following epochs. The training error, on the other hand, continues decrease. In epoch 2000, for EVA-b the training error amounts to 0.015, while the test error amounts to 0.031 (initial value: 0.11). For EVA-c the training error amounts to 0.021, while the test error amounts to 0.028 (initial value: 0.13). While further decrease of the training error may be expected from the diagrams, the generalization capabilities of EVA-b and EVA-c are unsatisfactory in comparison to EVA-a. The ratio of the number of free network weights to the number of training patterns is about 0.3 for both EVA-b and EVA-c. This is meaningful with regard to the goal of better generalization performance, while convergence of the training error is already fine.

The difference in the course of training for EVA-a on the one hand, and for EVA-b and EVA-c on the other hand, may be explained by the recurrent characteristics of the latter networks. The chronological patterns that are desired for the output of EVA-b and EVA-c are probably difficult to generalize. Nevertheless, as section 6.1 will show, the quality of the output obtained by EVA-b and EVA-c is sufficient to allow good results in model fitting.

5 Model Fitting

5.1 Obtaining Empirical Data from Real Human Subjects

For model fitting, empirical data from real human subjects is necessary. This data was obtained through a study at the Department of Psychology (University of Heidelberg)[23]. A group of 45 participants, mostly students, took part in this study. Each participant was seated in front of a computer screen, where he had to work on predefined PAD tasks 4 and 5 in the easiest mode of presentation (for the specifications of PAD tasks 4 and 5, see table 5.1/5.2; for the instructions of these PAD tasks, see appendix A). The overall time limit was set to 30 minutes. PAD task 4, which was presented first, had to be finished within 15 minutes.

The results of this study are presented in section 6.1 in direct comparison to the results obtained from subjects that are simulated by EVA.

Table 5.1: PAD task 4

Scheduled appointments at...	Earliest task starting time	Latest task starting time	Task duration	Priority
Storehouse	10:00 a.m.	0.15 p.m.	10 min.	high
Secretary's office	11.00 a.m.	4:00 p.m.	10 min.	normal
Administration	10:00 a.m.	2:30 p.m.	90 min.	very high
Printing office	10:00 a.m.	4:00 p.m.	90 min.	very high
Conference room	1:00 p.m.	1:00 p.m.	60 min.	very high

Optimal plan:
Move-to-administration → *Drive-by-car* → *Move-to-storehouse* → *Move-to-conference-room* → *Wait* → *Move-to-secretary's-office* → *Move-to-printing-office* → *Finish*

Table 5.2: PAD task 5

Scheduled appointments at...	Earliest task starting time	Latest task starting time	Task duration	Priority
Café	1:30 p.m.	2.30 p.m.	30 min.	high
Office	11.00 a.m.	2:00 p.m.	60 min.	very high
Administration	10:00 a.m.	4:00 p.m.	55 min.	high
Printing office	10:00 a.m.	3:00 p.m.	10 min.	normal
Conference room	11:30 a.m.	11:30 a.m.	45 min.	high
Central office	10:00 a.m.	4:15 p.m.	10 min.	very high

Optimal plan:
Move-to-printing-office → *Move-to-conference-room* → *Wait* → *Drive-by-car* → *Move-to-office* → *Move-to-café* → *Move-to-central-office* → *Move-to-administration* → *Finish*

[23] The study was carried out by Stefanie Nellen (University of Heidelberg), who gave her kind allowance to use her data in this thesis.

5.2 Simulating Subjects with EVA

After EVA's sub-networks were obtained through training, they were integrated into the PAD simulation system. The PAD simulation system is based on a C++ function library, which provides the necessary functions to load, create, and represent PAD tasks, and to represent and modify the current state of the PAD world. Furthermore, some functions serve to generate the input of EVA's sub-networks. Functions from the network library (see section 4.5.3) are used to load EVA's sub-networks into the PAD simulation system and to use them for the transformation of input to output patterns. Specialized functions are applied for analyzing and evaluating the actual course of planning and to write the results presented in section 6 into log files. Thus, by employing the PAD simulation system EVA can be manipulated to simulate subjects working on PAD tasks. Since the human subjects that serve as standard of comparison worked on predefined PAD tasks 4 and 5, in the course of model fitting the simulation runs are also restricted to these PAD tasks.

5.3 Accessible Parameters in Model Fitting

Model fitting has the goal to adjust free model parameters in a way, that the "behavior" of the model is as similar as possible to the behavior of real human subjects. Naturally, the comparison of behavior is restricted to the subset of human behavior that the model claims to explain. EVA was developed to model the sequential use of the 13 available PAD operators in solving a PAD task. As pointed out in section 4.1, the objective in doing so was not to precisely predict an individual course of planning for a single human subject, but to obtain a good model fit on the level of whole samples.

Before model fitting, two questions must be addressed. First, by which indicators should the human sample and the simulated sample be compared? Second, which model parameters can be varied to obtain a better model fit? For EVA, these questions can be addressed more straightforwardly, when one differentiates between two stages of model fitting.

5.3.1 First Stage of Model Fitting

The first stage of model fitting may be identified to a large extent with model development in general. After the basic framework of EVA was laid down, the simulated data was continuously compared with the empirical data obtained by the study described in section 5.1. This comparison was carried out with regard to many of the indicators described in the results section (section 6). To avoid too much redundancy, I omit a specification of these indicators at this point. It became clear, that many of the assessed indicators could not be influenced by varying model parameters, but resulted from a complex interaction of the global model architecture, of the structure of the training data, and even of the progress in network training (therefore, one

may call them “emergent indicators”). Other indicators could be influenced at least indirectly. For example, as it became clear, that EVA had problems with the optimum use of the drive by car, the training data was modified to give EVA more learning experience in this area. Later on, when it became obvious, that EVA needed an external memory aid, because the recurrent connections were not sufficient for differentiated remembering, the external situation-action memory was added to the input. Furthermore, to improve the end score EVA achieved, the number of network units and training patterns was increased.

Thus, in this basic stage of model fitting not only were parameter values adjusted, but also the specification of the model, its overall structure, its components, and the inner architecture of its sub-networks were changed. Of course, not only the comparison to empirical data, but also technical and theoretical considerations influenced the specification of the model and its (free) parameters. Most of these considerations were mentioned in section 4 in the respective passages. Table 5.3 provides an overview of the model specifications and model parameters influenced through the comparison of empirical and simulated data. However, it would exceed the size of this thesis to mention every adjustment that was carried out on the way from the first sketch of EVA to its current specification and parameter settings.

In addition, pre-experiments with MPAD (“Mini-Plan-A-Day”) were important, especially with respect to the topology of EVA’s sub-networks and to their training. Furthermore, some parameter adjustments are dependent upon each other. For example, for improving EVA’s end score in PAD tasks 4 and 5, the number of training patterns was increased. However, as pointed out in section 4.5.4, the number of network weights partly depends on the number of training

Table 5.3: Model specifications and model parameters influenced through the first stage of model fitting

Subject of specification / Parameter	Section(s) addressing the specification or parameter
Subjects of specification	
Input representation	4.2.3, 4.3.1
Subdivision of EVA in sub-networks	4.2.4, 4.5.2.3
Topology of EVA’s sub-networks • Layer size • Shortcut connections • Recurrent connections	4.4
Parameters	
MemForgetFactor (for the training input)	4.3.1, 4.5.2.2
RestrainFactor	4.3.2.2
→ Training data parameters in table 4.5	4.5.2
→ Training data parameters in table 4.6 (4.7/4.8)	4.5.2
DelRepeatFactor	4.5.2.2 (4.4.2)
ExpET	4.5.2.3 (4.4.3)
EndThreshold	4.5.2.3 (4.4.3)
→ Number of training patterns	4.5.2.2, 4.5.2.3, 4.5.4

patterns. Therefore, in enlarging the training set one must also enlarge the size of the hidden network layers or the number of shortcut connections.

After all, the most important measure for the quality of EVA was its performance, indicated by the average end score and max score, and by the proportion of final plans identical to the optimum plan. Nevertheless, the performance alone will not lead to a good model fit, if all other indicators (presented in section 6.1) are different for simulated and human samples.

5.3.2 Second Stage of Model Fitting

The second stage of model fitting began, after EVA was specified in its current shape and the training of the actual instances of EVA's sub-networks were completed. Then, there are still seven Parameters left which can be adjusted to reach an optimum fit between empirical and simulated data. These parameters are: NoiseTime, NoiseFlag, NoisePriority, NoiseProportion, $\text{MemForgetFactor}_{\text{Simul}}$, f_D, and f_F (each of which can be varied in the range [0.0; 1.0]). The first four parameters are used to control the noise that is applied to EVA's input. NoiseProportion determines the maximum proportion of input units that is disturbed. For example, a value of 0.4 for NoiseProportion implies that in every processing cycle between 0% and 40% of the input units are disturbed. When an input unit is (randomly) chosen for distortion, then it is altered by the product of a random value between 0.0 and 1.0 and its respective noise factor. The noise factors can take the values NoiseTime, NoiseFlag, and NoisePriority. The assignments of input units to noise factors are shown in table 4.4 (NoiseTime = 0.01, NoisePriority = 0.025, NoiseFlag = 0.05). These assignments are based on the consideration that the "information density" differs between input units. Input units like <Time1> or <Time2> have a high density, since an activation difference of, e.g., 0.1 represents almost a whole hour difference. On the other hand, for an input unit like <CarUsed> an activation difference of 0.1 is only a slight disturbance. Since the values for the noise factor parameters are based on technical considerations, they were not varied in the process of model fitting. For this reason, they were omitted in the following.

$\text{MemForgetFactor}_{\text{Simul}}$ is the MemForgetFactor used for the external situation-action-memory in the simulation runs. As the value 0.7, which is used for the generation of the training data, may not be optimal for the reproduction of empirical data, it may be meaningful to chose a different value for $\text{MemForgetFactor}_{\text{Simul}}$. f_D and f_F are the factors by which the output units of EVA-b and EVA-c, respectively, are multiplied, before the output unit with maximum activation is determined (see section 4.2.4). Therefore, by f_D the Delete evaluator can be weakened or strengthened, and by f_F the Finish evaluator can be weakened or strengthened. These factors are meaningful, since due to the separate training of EVA-a, EVA-b, and EVA-c, their output units may operate on slightly different scales.

Thus far, I considered the parameters accessible for adjustment in the second stage of model fitting. However, the indicators which were considered in the process of model fitting must also be stated. These indicators are described in more detail in section 6.1. To avoid

redundancy, only a brief list is given at this point. The following indicators were taken into consideration for model fitting:

- The end score and max score, both in the standard evaluation and the advanced evaluation
- The proportion of planning processes for which the end score and max score are equal to the optimal score, respectively
- The overall length of the sequences of operators
- The number of complete deletions of the previous generated plan in the overall planning process

For model fitting, no quantitative method was used. Instead, the parameter settings were varied manually. In doing so, the indicator differences between the simulated and the empirical sample were observed. The goal was to find a parameter configuration, for which these differences are as small as possible. The overall length of the sequences of operators had the highest priority, as the simulated and the empirical sample can only be compared meaningfully, if at least this length is in the same order of magnitude. For the other indicators, an attempt was made to distribute the remaining differences as evenly as possible.

Unfortunately, regarding the stated indicators, it was not possible to find a common parameter configuration that allowed a good fit for both PAD task 4 and PAD task 5. Instead, two different parameter configurations were found:

- For PAD task 4: $f_D = 0.8$, $f_F = 0.75$, $\mathsf{NoiseProportion} = 0.1$, $\mathsf{MemForgetFactor}_{\mathsf{Simul}} = 0.8$
- For PAD task 5: $f_D = 0.8$, $f_F = 0.42$, $\mathsf{NoiseProportion} = 0.3$, $\mathsf{MemForgetFactor}_{\mathsf{Simul}} = 0.8$

Only f_D and $\mathsf{MemForgetFactor}_{\mathsf{Simul}}$ are constant between both configurations. The differences of the other parameters are discussed in section 7.1.

6 Results

This section presents results from various simulation runs. First, in section 6.1, the planning behavior of real and simulated subjects in PAD tasks 4 and 5 is compared (as direct outcome of model fitting). Second, in section 6.2, EVA is applied to randomly generated PAD tasks of different complexity. Third, in section 6.3, a first attempt at validation is presented.

6.1 Results of Model Fitting

In the following, a sample of subjects simulated by EVA and a sample of human subjects are compared to each other. The sample of human subjects was assessed in the study that is described in section 5.1. Both human and simulated subjects had to work on PAD tasks 4 and 5. The human sample comprises 45 subjects, while the simulated sample comprises 4500 subjects. By such a large number of simulation runs the results obtained by EVA are highly reliable. Since no inference statistical tests are planned, the different sample size does not cause any problem.

As stated in section 5.3.2, in fitting the results obtained by EVA for PAD tasks 4 and 5 to the results obtained by the human sample, the following parameter configurations were found out for EVA:

- For PAD task 4: $f_D = 0.8$, $f_F = 0.75$, $\mathrm{NoiseProportion} = 0.1$, $\mathrm{MemForgetFactor}_{\mathrm{Simul}} = 0.8$
- For PAD task 5: $f_D = 0.8$, $f_F = 0.42$, $\mathrm{NoiseProportion} = 0.3$, $\mathrm{MemForgetFactor}_{\mathrm{Simul}} = 0.8$

The simulated data presented in this section (6.1) is based on these parameter configurations. In appendix C, one can find the data sheets that were generated by the PAD simulation system for evaluation purposes. Many of the variable values presented in the following are taken from these sheets.

6.1.1 Performance

There exist several variables that serve as performance measures. First, there are the end score and max score, as described in section 3.1.2. They are calculated both in the standard way and in the advanced way. In the advanced evaluation, visits made without carrying out an appointment are punished by subtracting five points from the score. As stated in section 4.3.2.1, the training data for EVA's sub-networks is based on this evaluation. The corresponding variables are called `EndScoreStd`, `MaxScoreStd`, `EndScoreAdv`, `MaxScoreAdv`. For comparison: The optimal score amounts to 28 points for PAD task 4 and to 26 points for PAD task 5.

Furthermore, an interesting measure of performance is the proportion of subjects, which found out the optimal plan. On the one hand, the end score may be identical with the optimal score, or on the other hand, only the max score is identical with the optimal score, because the

Table 6.1: Comparison of performance indicators for human and simulated subjects. The optimal score amounts to 28 points for PAD task 4 and to 26 points for PAD task 5.

	PAD task			
	Nº 4		Nº 5	
	Sample		Sample	
	Humans	EVA	Humans	EVA
EndScoreStd	25.3 (5.1)	21.2 (7.4)	22.9 (3.7)	22.1 (3.1)
MaxScoreStd	26.5 (2.1)	25.5 (2.6)	23.7 (2.6)	22.9 (1.4)
EndScoreAdv	22.2 (5.7)	21.2 (7.4)	21.9 (4.7)	22.1 (3.2)
MaxScoreAdv	26.4 (2.6)	25.5 (2.6)	23.6 (2.7)	22.9 (1.4)
ESOptRatio	22.2%	22.8%	37.8%	6.3%
MSOptRatio	24.4%	23.5%	40.0%	7.0%

optimal plan was lost in further planning. Thus, there are two different variables: ESOptRatio and MSOptRatio (naturally, ESOptRatio $\leq$ MSOptRatio). They are also given as percentages in relation to the overall number of subjects. Because there are no differences in these variables due to standard vs. advanced evaluation, no further distinctions are made.

EndScoreStd, MaxScoreStd, EndScoreAdv, MaxScoreAdv, ESOptRatio, and MSOptRatio were considered both in the first and second stage of model fitting. Their values are presented in table 6.1. While for most of these indicators the differences between EVA and the human sample are only small, large differences remain for PAD task 5 for ESOptRatio and MSOptRatio. Evidently, for EVA it is much more difficult to find the optimal plan for PAD task 5 than for human subjects to do so. However, in PAD task 4 EVA reaches the optimal plan as often as the human subjects. On the other hand, EndScoreStd for PAD task 4 is four points less for EVA than for the human subjects. For EndScoreAdv this difference only amounts to one point. This shows, that human subjects more often make senseless visits in their final plans than EVA does. During training, EVA has learned to avoid such visits.

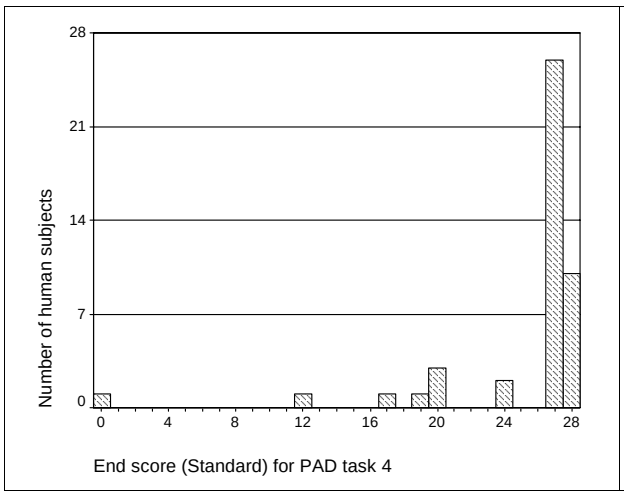

Fig. 6.1: Distribution of EndScoreStd for PAD task 4 for the human sample

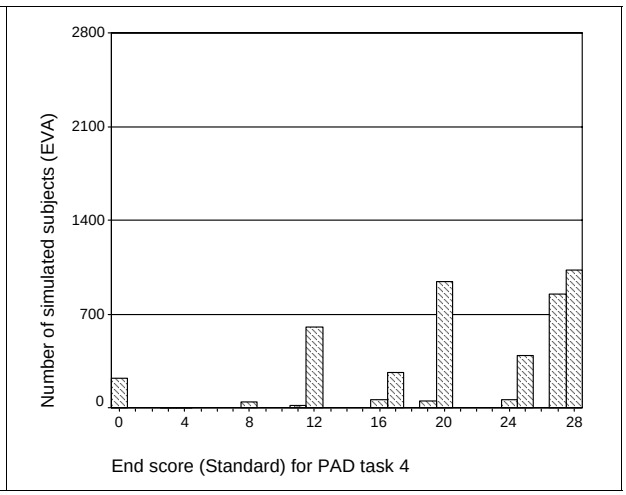

Fig. 6.2: Distribution of EndScoreStd for PAD task 4 for EVA

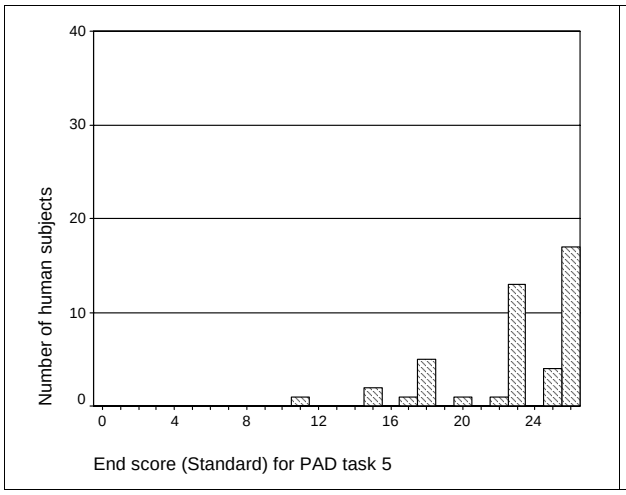

Fig. 6.3: Distribution of EndScoreStd for PAD task 5 for the human sample

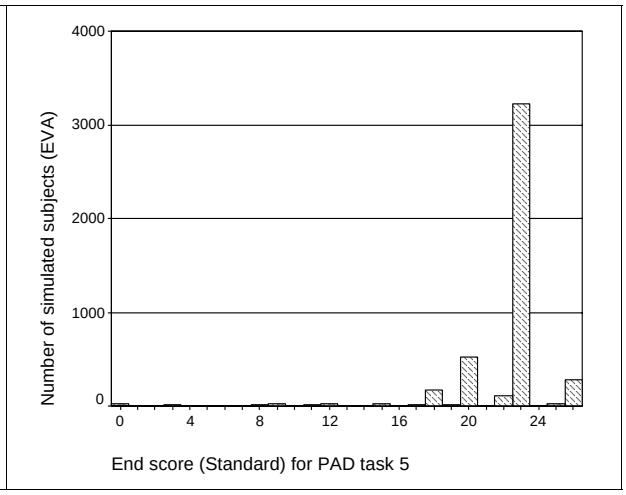

Fig. 6.4: Distribution of EndScoreStd for PAD task 5 for EVA

Furthermore, not only the means and standard deviations, but also the distributions of `EndScoreStd` were compared. They are shown in figure 6.1 to 6.4. For PAD task 4 (fig. 6.1/6.2) the comparison reveals, that the distributions are quite different. For the human sample there is a clear peak at 27 points, while for EVA the distribution is substantially more even. However, at least the three end scores with the highest frequency are equal for the human sample and EVA: 20, 27, and 28 points. For PAD task 5, the comparison leads to the opposite result: The strong peak that EVA produces at 23 points appears in the distribution of the human sample only in weakened form. Since the comparison of frequency distributions even on the level of end scores yields such differences, a further distinction of final plans was not considered.

6.1.2 Overt Characteristics of the Planning Process

One of the most important overt characteristics of the planning process is the overall length of the sequence of operators generated in the planning process. This variable is called `Length` and gives the number of applied operators. Furthermore, the average number of successive *Delete* operators (`LenDelChain`) and the number of complete deletions of the previously generated plan in the overall planning process (`NumTotalDel`) are interesting. The mean values of these three

Table 6.2: Comparison of outer characteristics of the planning process for human and simulated subjects

	PAD task			
	Nº 4		Nº 5	
	Sample		Sample	
	Humans	EVA	Humans	EVA
`Length`	40.6 (30.4)	40.5 (29.5)	43.6 (27.6)	44.9 (41.2)
`LenDelChain`	2.2 (1.3)	2.7 (1.2)	2.2 (1.5)	2.5 (1.1)
`NumTotalDel`	2.9 (3.1)	3.0 (3.1)	2.9 (2.6)	2.0 (2.7)

variables are given in table 6.2. In addition, the distribution of `Length` is shown in figure 6.5 to 6.8, the distribution of `NumTotalDel` in figure 6.9 to 6.12. For the optimal plans, the number of applied operators (actions plus *Finish*) is eight for PAD task 4 and nine for PAD task 5.

The mean values of `Length` and `NumTotalDel` were used as indicators for the second stage of model fitting. For EVA, `Length` can be increased quite directly by decreasing the value of f_F. Therefore, it was possible to obtain such a good fit with regard to this indicator both for PAD task 4 and 5, as table 6.2 reveals. `NumTotalDel` is mainly dependent on the value of f_D (increasing f_D leads to an increasing `NumTotalDel`). However, in keeping f_D constant between PAD tasks 4 and 5, it was not possible to obtain an equally good fit for both PAD tasks. In addition, f_D also influences `LenDelChain` in the same direction as `NumTotalDel`, and for PAD task 5 `LenDelChain` is even higher for EVA than for the human sample. Therefore, an increase of f_D would lead to a better fit of PAD task 5 for `NumTotalDel`, but to an even worse fit for `LenDelChain`.

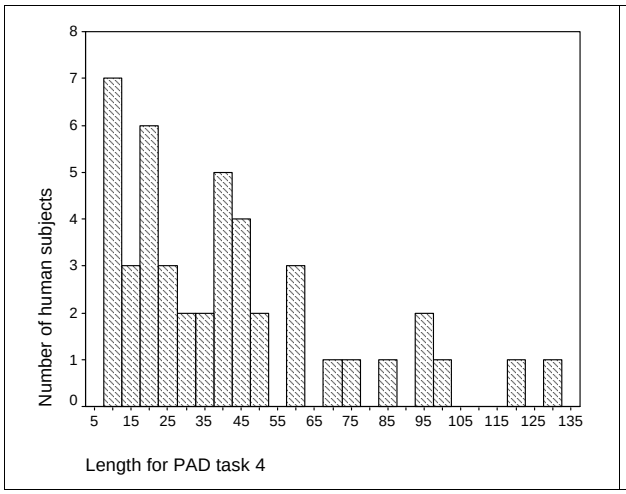

Fig. 6.5: Length of the sequence of operators for PAD task 4 for the human sample

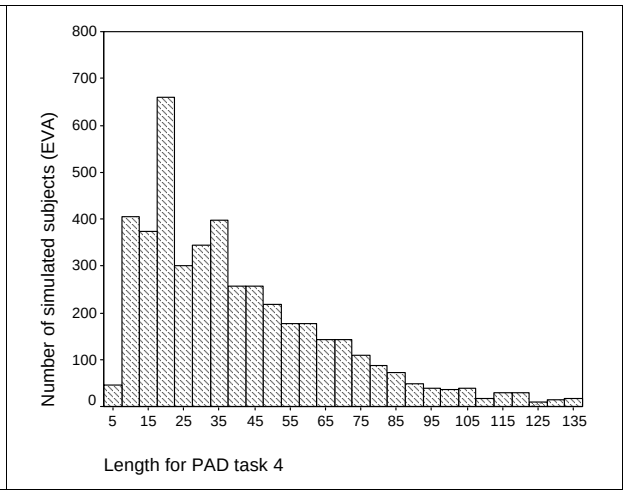

Fig. 6.6: Length of the sequence of operators for PAD task 4 for EVA (Maximum of Length at 208)

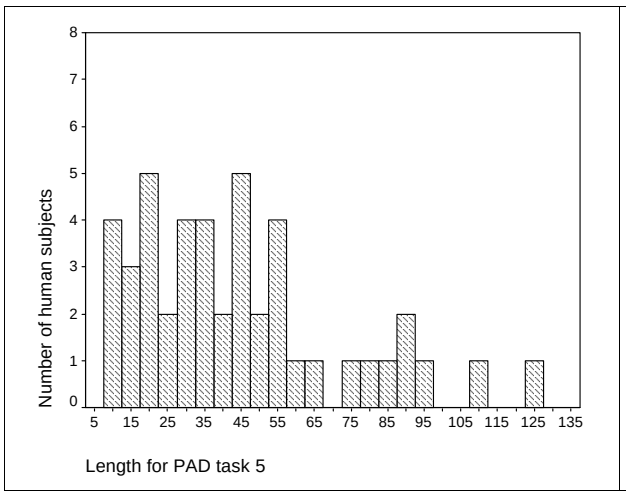

Fig. 6.7: Length of the sequence of operators for PAD task 5 for the human sample

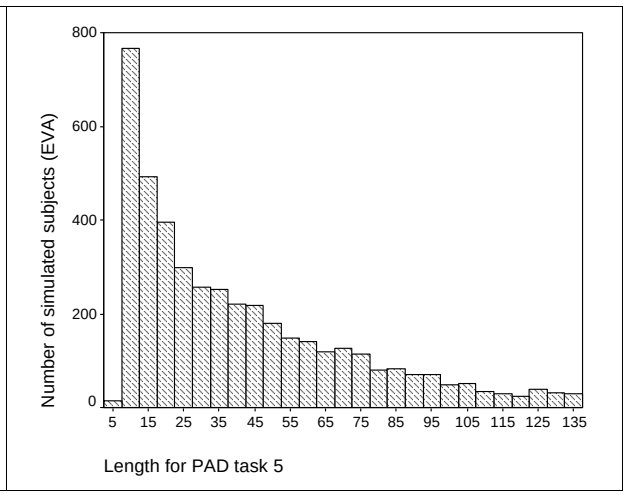

Fig. 6.8: Length of the sequence of operators for PAD task 5 for EVA (Maximum of Length at 393)

In the first stage of model fitting, it became obvious, that EVA-b needed recurrent connections, since the delete chains produced by EVA so far were only slightly larger than 1.0 on average. Thus, only by changing some of the basic specifications of EVA, it was later possible to obtain a good fit through manipulating f_D.

The comparison of the distributions of `Length` shown in fig. 6.5 to 6.8 reveals, that for PAD task 4 even on the level of the `length` distribution a quite good fit is noticeable. On the other hand, for PAD task 5 the shape of the distribution of the human sample is really different to the shape of the distribution of the simulated sample. The same pattern appears in the distributions of `NumTotalDel` (fig. 6.9 to 6.12). The fit for PAD task 4 is remarkably good (even the minimum at `NumTotalDel` = 5 appears in both the distribution of the human sample and of EVA), while the fit for PAD task 5 is not satisfying. The tendency of EVA to perform only one total deletion is too strong in comparison to the human sample.

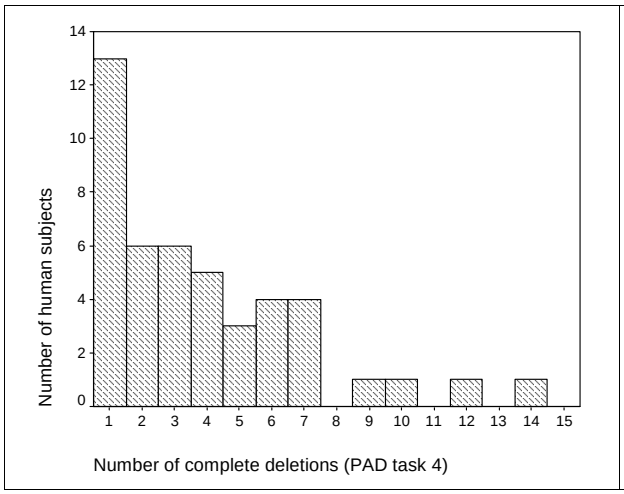

Fig. 6.9: Distribution of NumTotalDel for PAD task 4 for the human sample

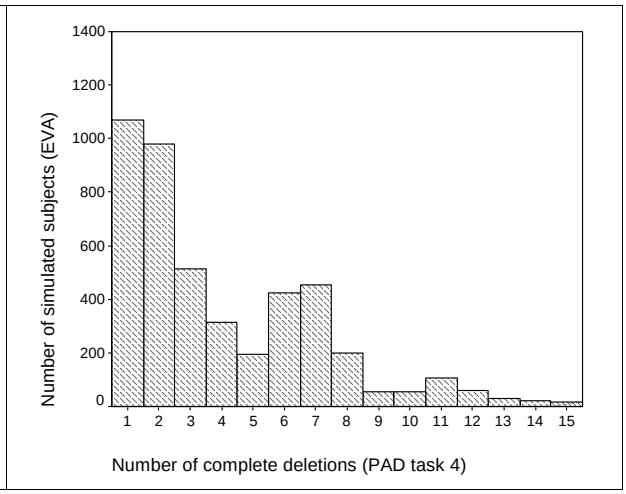

Fig. 6.10: Distribution of NumTotalDel for PAD task 4 for EVA (Maximum of NumTotalDel at 21)

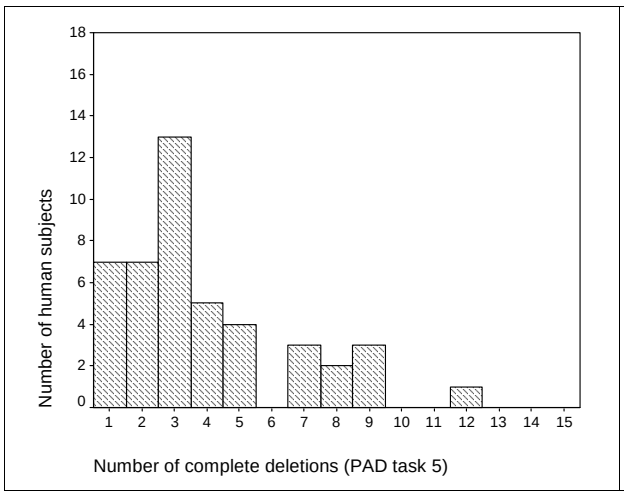

Fig. 6.11: Distribution of NumTotalDel for PAD task 5 for the human sample

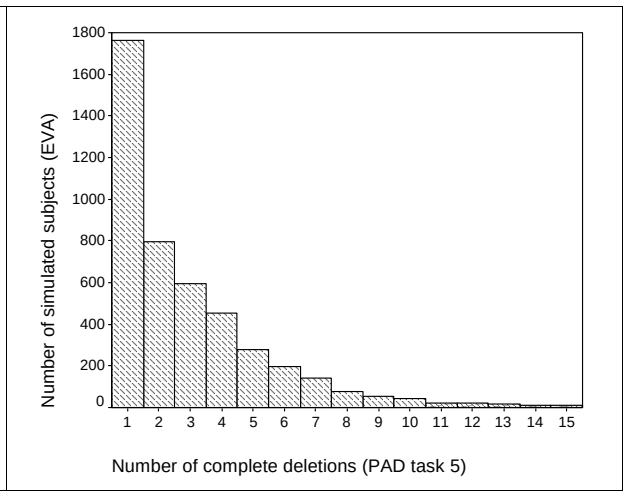

Fig. 6.12: Distribution of NumTotalDel for PAD task 5 for EVA (Maximum of NumTotalDel at 27)

Thus, the comparison of `Length`, `NumTotalDel`, and `LenDelChain` reveals, that especially for PAD task 4, a good fit even on the level of frequency distributions could be reached, while the fit for PAD task 5 remains unsatisfactory.

6.1.3 Operator Use

In this section, a comparison of the frequency of operator use is provided. Fig. 6.13 shows the proportion of the use of each operator for PAD task 4 relative to the total number of applied operators. Differences larger than 2% between the human and the simulated sample must be stated for the operators *Move-to-storehouse*, *Move-to-administration*, *Wait*, and *Delete*. However, altogether the frequency patterns are quite similar to each other. The high frequency of the *Delete* operator shows an especially similarity in the same order of magnitude for both samples. Furthermore, the order of rank for the *Drive-by-car* and *Wait* operator is met by EVA. Taken together, the similarities between the human and the simulated sample are remarkable, especially in light of the fact, that the parameter adjustments in the second stage of model fitting were not aimed at this frequency profile. Furthermore, even if the increase of f_D increases the frequency of use of the *Delete* operator ($frequ_{Del}$), the order of magnitude of this frequency is relatively constant for meaningful values of f_D when keeping the remaining parameter settings at their standard values (f_D = 0.4 [n=450] $\Rightarrow$ $frequ_{Del}$ = 33.2%; f_D = 0.8 [std., n=4500] $\Rightarrow$

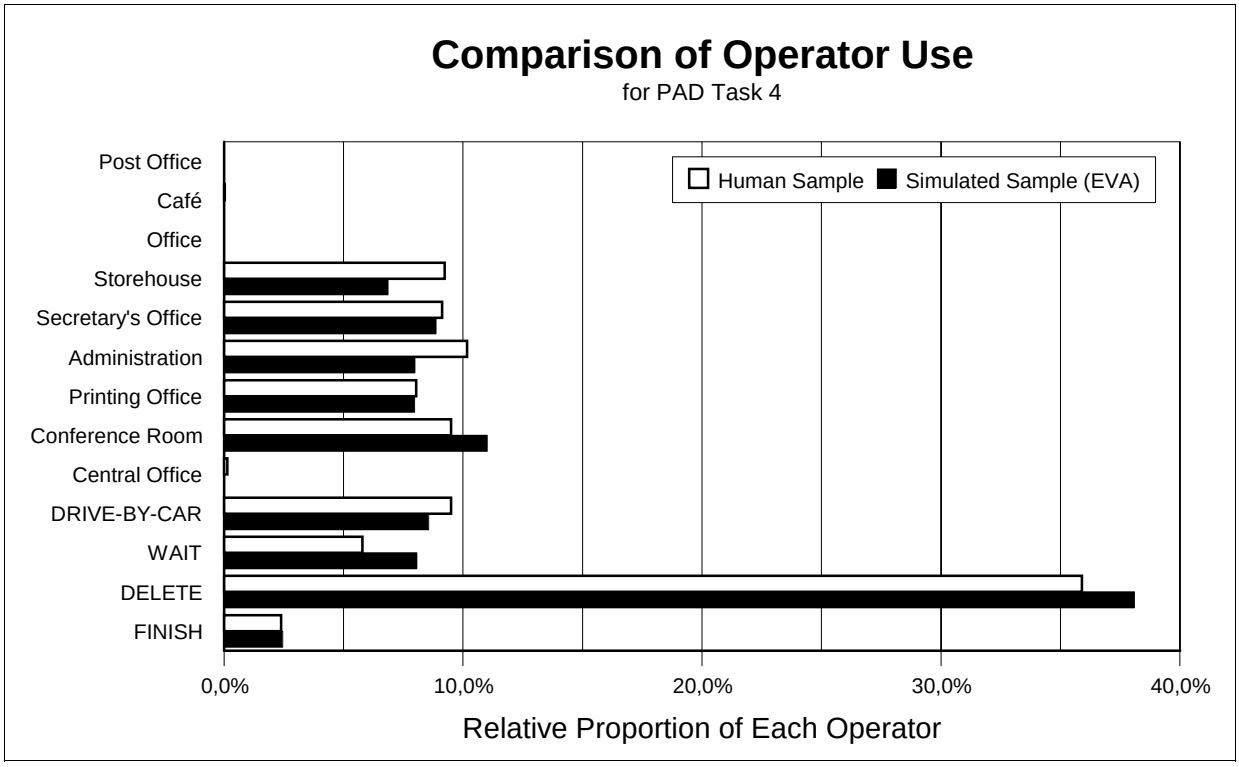

Fig. 6.13: Comparison of operator use for PAD task 4. White bars indicate the relative proportion of each operator for the human sample, black bars indicate it for the sample simulated by EVA. Movement operators are represented by their goals of movement.

$frequ_{Del}$ = 38.1%; f_D = 1.0 [n=450] ⇒ $frequ_{Del}$ = 42.7%; for the human sample [n=45]: frequDel = 35.9%).

For PAD task 5 (see fig. 6.14), the comparison between the human and the simulated sample also reveals quite a good fit with regard to the frequency profile. Only three differences are larger than 2%, namely for *Move-to-café*, *Move-to-conference-room*, and *Delete*. On the other hand, the difference for *Move-to-conference-room* amounts to 4.6%, which is very much in relation to the absolute size of the frequency (8.2% for the human sample, 3.6% for the simulated sample). Obviously, the tendency to move to the conference room is much smaller for EVA than it is for the human subjects.

To sum up, with regard to the relative frequency of operator use, EVA is able to reproduce the data obtained by the human sample. Since the second stage of model fitting was not aimed at this frequency profile, the correspondence emerges from the basic architecture of EVA and from the structure of the training data for EVA's sub-networks.

6.1.4 Use of Heuristics

In their original work about PAD, Funke and Krüger (1995) present eight heuristics, which participants may use when working on PAD tasks. They have implemented five of these heuristics in a tool for the evaluation of planning processes. Since such an evaluation is very interesting

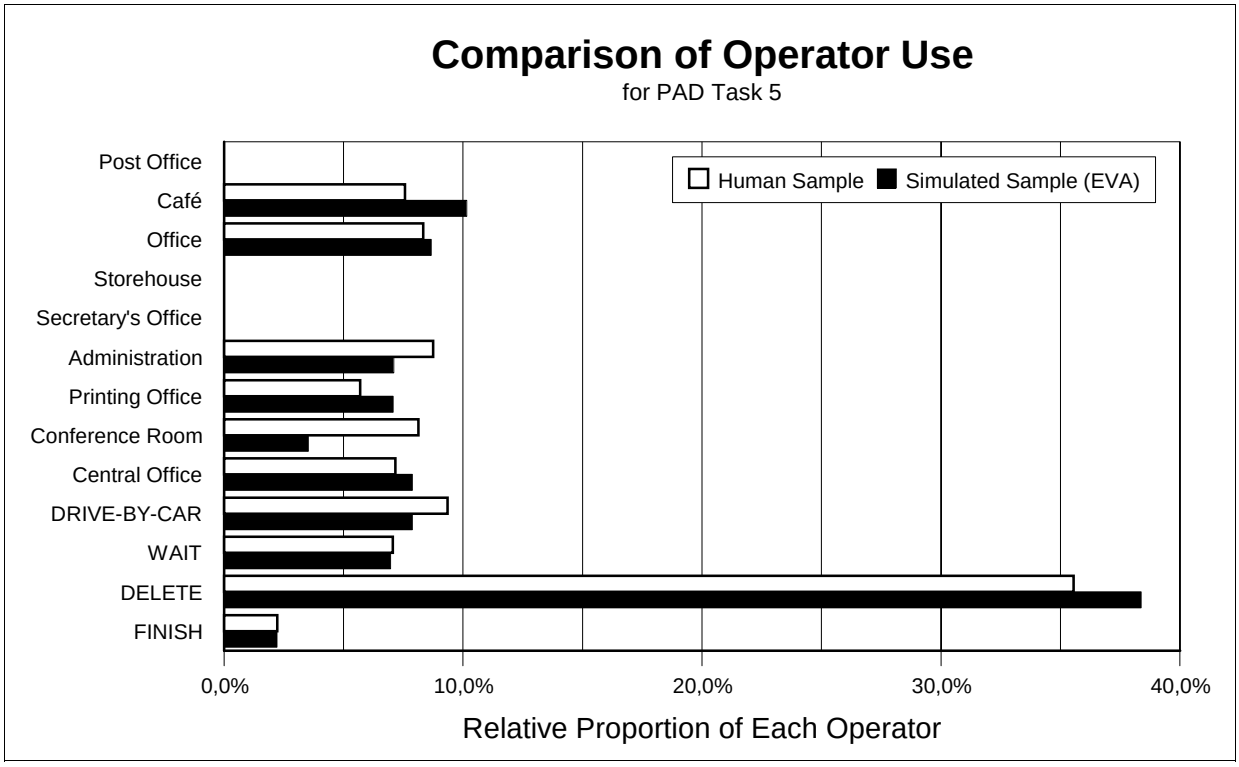

Fig. 6.14: Comparison of operator use for PAD task 5. White bars indicate the relative proportion of each operator for the human sample, black bars indicate it for the sample simulated by EVA. Movement operators are represented by their goals of movement.

Table 6.3: Comparison of heuristic application. The meaning of the heuristics is explained in the text.

	PAD task			
	Nº 4		Nº 5	
	Sample		Sample	
	Humans	EVA	Humans	EVA
Minimizing movement times				
Performance	1.90 (0.30)	2.04 (0.26)	2.16 (0.41)	2.11 (0.27)
Random	2.13 (0.17)	2.14 (0.19)	2.39 (0.16)	2.25 (0.17)
Maximizing the advantage of the drive by car				
Performance	1.96 (0.89)	1.62 (0.53)	1.81 (0.59)	2.00 (0.51)
Random	2.12 (0.46)	2.10 (0.35)	2.56 (0.34)	2.42 (0.29)
Considering priority				
Performance	1.74 (0.25)	1.74 (0.19)	1.97 (0.30)	1.86 (0.23)
Random	1.74 (0.17)	1.78 (0.16)	2.01 (0.13)	1.87 (0.13)
Considering urgency				
Performance	1.71 (0.27)	1.74 (0.29)	2.06 (0.46)	2.24 (0.43)
Random	1.97 (0.17)	1.98 (0.20)	2.41 (0.15)	2.27 (0.17)
Minimizing waiting time				
Performance	1.40 (0.17)	1.51 (0.24)	1.54 (0.25)	1.41 (0.20)
Random	1.59 (0.12)	1.58 (0.12)	1.68 (0.14)	1.47 (0.14)

with regard to EVA, these five heuristics were also integrated into the evaluation procedures of the PAD simulation system. In the following, the five heuristics are briefly presented:

- "Minimizing movement times": Go to the location which is the nearest one (with the least necessary movement time)
- "Maximizing the advantage of the drive by car": If you drive by car, then go to the location which is farthest away.
- "Considering priority": Go to a location with an appointment of the highest existing priority.
- "Considering urgency": Fulfill the appointments in the order of their latest time for task starting.
- "Minimizing waiting time": Go to the location where you must wait the least time for task starting.

For each subject (or process of planning) the following analysis is carried out: Each heuristic is evaluated for every situation that the subject encounters in the PAD world and in which the subject applies a *Movement* action, in two respects. First, the *Movement* actions that correspond to locations in the appointment list of the PAD task are put in a ranking order due to the respective heuristic. For example, for the "Minimizing movement times" heuristic, the *Movement* action corresponding to the nearest location receives a value of one, the *Movement* action corresponding to the second nearest location receives a value of two, and so forth. In doing so, ranking

positions can be shared; then the following ranking position(s) is/are omitted. Afterwards, the rank of the *Movement* action the subject actually chose is recorded (called "performance value"). In addition, a second value is recorded, namely the average of all allocated ranks. This average is equivalent to the mean result of a random choice of a *Movement* action (therefore it is called "random value"). Thus, a performance and a random value are recorded for each of the five heuristics and for every situation that the subject encounters in the PAD world and in which he applies a *Movement* action. These values are averaged out, so that one average performance and one average random value exist for each heuristic. These average values are the ten heuristic indicators for each subject. For each heuristic, through comparing the random value to the performance value, one can determine, if the subject was following the heuristic or not. If the subject applied the heuristic to his movement decisions, the performance value must be smaller than the random value. Otherwise, the respective heuristic was not taken into account.

Table 6.3 shows the ten mean heuristic indicators for both PAD tasks 4 and 5 and both for the human and the simulated sample. As an inspection of the table reveals, for most of the heuristics, EVA follows the same tendency as the human subjects, even if the size of the difference between the random and the performance value is sometimes different.[24] For PAD task 4, the goodness of fit is remarkably high. As the human subjects, EVA applies all heuristics except for "Considering priority". "Maximizing the advantage of the drive by car" is even more pronounced by EVA than it is by the human subjects. On the other hand, for the other applied heuristics, the differences between the random and performance values are smaller for EVA than for the human subjects. For PAD task 5, the comparison between EVA and the human sample reveals one difference: The heuristic "Considering urgency" is ignored by EVA, while it is applied by the human subjects. As for PAD task 4, the particularly small difference between the random and the performance value for the heuristic "Considering priority" is reproduced by EVA.

The good fit between the empirical and simulated data, with regard to heuristic application is considerable, since the application of heuristics was not subject of model fitting in general, neither in the first nor in the second stage. Nevertheless, EVA's heuristic application is probably influenced by the structure of the training data. For example, previous versions of EVA ignored the heuristic "Maximizing the advantage of the drive by car". In their training data, half of the randomly generated PAD tasks had optimal plans without the application of *Drive-by-car*. Only when the training data was changed, so that about 85% of the optimal plans included a drive by car (see sections 4.5.2.1/4.5.2.2), did EVA begin to apply the car advantage heuristic. Furthermore, while previous versions of EVA showed smaller performance values for "Considering priority", this value might be increased to a random level by setting the priorities of all appointments to equal levels in every ninth PAD task of the current training data for EVA-a/-b (see section 4.5.2.2). However, the latter remark is purely speculative. Unfortunately, the large amount of time needed to train new instances of EVA-a and EVA-b for ruling out this possibility is no longer available. Therefore, this remark must be stated to prevent from an

[24] The following boundary is set arbitrarily: A heuristic is categorized as "applied", if its performance value is at least 0.05 less than its random value.

overestimation of the properties just emerging from EVA's basic architecture. In order to correctly reproduce the application of heuristics, EVA must be at least partly trained with adequate data.

6.1.5 Arriving at Locations

When a subject arrives at a location with a scheduled and unfulfilled appointment, there are four possibilities: (1) The subject is too late for task starting. (2) The subject is on time and can begin immediately with task fulfillment. (3) The subject is too early, but does not wait. (4) The subject is too early and waits until the earliest time for task starting. For each course of planning, the relative proportions of each of the four possibilities are calculated; they correspond to the variables `PropTooLate`, `PropOnTime`, `PropTooEarly`, and `PropTooEarlyWait`, the means and standard deviations of which are given as percentages in table 6.4 for both PAD task 4 and 5 and for both the human and the simulated sample.

Further, the average values for the periods during which the subjects were too early (`TimeTooLate`) and for the periods in which they were too late (`TimeTooEarly`) are shown in table 6.4. These averages are based on the arrivals upon which the subjects actually were too late or too early.

With regard to the proportion of different arrival types, EVA fits the human data quite well (again, especially for PAD task 4). For each of the four arrival indicators, the deviations between human and simulated data are small in relation to the absolute magnitude of the proportion values. For PAD task 5, the fit is a little worse, since the differences in PropTooLate (Humans: 11.7%; EVA: 18.3%) and PropTooEarly (Humans: 3.0%; EVA: 1.4%) are quite large in relation to the absolute magnitude of these proportion values. However, EVA meets the fact, that `PropOnTime` is larger for PAD task 5 than for PAD task 4 for the human sample (70.3% to 60.9%; EVA: 67.8% to 62.5%).

Table 6.4: Arriving at locations. The meaning of the variables is explained in the text.

		PAD task			
		Nº 4		Nº 5	
		Sample		Sample	
		Humans	EVA	Humans	EVA
Proportion of arrivals					
`PropTooLate`	[%]	21.4 (7.9)	17.4 (6.9)	11.7 (8.5)	18.3 (10.3)
`PropOnTime`	[%]	60.9 (7.4)	62.5 (8.5)	70.3 (11.8)	67.8 (9.1)
`PropTooEarly`	[%]	3.5 (4.6)	3.5 (5.0)	3.0 (4.1)	1.4 (3.4)
`PropTooEarlyWait`	[%]	14.2 (5.7)	16.6 (6.3)	15.0 (7.7)	12.5 (6.4)
Periods being too late/too early					
`TimeTooLate`	[min.]	29.2 (17.8)	45.7 (31.7)	26.5 (24.8)	45.1 (28.0)
`TimeTooEarly`	[min.]	24.9 (20.2)	32.8 (21.1)	11.3 (8.2)	15.7 (11.6)

The periods of being too late or too early are larger for EVA than for the human sample. These differences are smaller for TimeTooEarly than for TimeTooLate. The values EVA produces between factor 1.3 (TimeTooEarly/PAD task 4) and factor 1.7 (TimeTooLate/PAD task 5) are larger than the values of the human sample. These factors do not seem to be remarkably high. In addition, one must take into consideration, that EVA gets time information on a scale from 0.1 to 1.0 (see table 4.4), which corresponds to a period of 8½ hours. In light of this fact, it is surprising, that a multi-layer network like EVA-a is able to determine the goals of movement with such precision. Normally, multi-layer networks tend to generalize over neighboring input values, but instead, EVA-a has to develop discrimination capabilities during training. These discrimination capabilities depend on the number of network units and training patterns. Forerunners of EVA, with fewer units and fewer training patterns showed substantially worse values for TimeTooEarly and TimeTooLate.

Apart from TimeTooEarly and TimeTooLate, the variables considered in this section were not subject to model fitting. Therefore, the goodness of fit of these variables mainly depends on the basic architecture of EVA and on the basic structure of the training data.

6.1.6 Course of Planning

In general, it is difficult to find indicators that grasp genuine characteristics of the planning process. By calculating mean values of global indicators as in the preceding sections, the view of the process of planning remains at a surface level. For this reason, this section presents three attempts to gain a deeper insight into the process of planning. These attempts comprise a qualitative approach, the presentation of an interesting correlation, and again, the consideration of operator frequency.

6.1.6.1 Examples for the Course of Planning

In this subsection two typical courses of planning are presented, one from a human subject and one from EVA (fig. 6.15/6.16). As pointed out before, EVA is not intended to simulate individual courses of planning for single subjects, but nevertheless this aspect is of importance, if EVA is to cover typical characteristics of the human planning process. Of course, such a presentation of single examples can only claim to have demonstrative character. The conclusions drawn from an inspection like this are purely qualitative and also very subjective. Nevertheless, such an approach has its legitimacy as a supplementation of the quantitative and global comparisons presented so far. If the reader trusts in the claim of the author, that these examples are actually “typical” with regard to overt characteristics recognizable by a short view into the log file from which they were taken, these examples may be worthy of taking note. They may provide a kind of evidence, that the global indicators compared so far were not caused by totally different courses of planning, the only similarity of which is the similarity of their indicator means. (I

Fig. 6.15: Example for the course of planning, performed by a human subject on PAD task 4 (printout from a log file generated by the PAD simulation system)

```
Sbj-No.: 39     (5 Locations) (OptRating: 28)
---------------------------------------------
Sto [1]:  10:00 - 12:15  (10)
Sec [0]:  11:00 - 16:00  (10)
Adm [2]:  10:00 - 14:30  (90)
Pri [2]:  10:00 - 16:00  (90)
Con [2]:  13:00 - 13:00  (60)

  1.  [Admi][+]  <- 8->  Ad** [ 8]
  2.  [Stor][+]  <- 3->  Ad** [ 8] -> Sto* [ 3]
  3.  [DEL ][+]  <- 8->  Ad** [ 8]
  4.  [DEL ][+]  <- 0->
  5.  [Stor][+]  <- 3->  Sto* [ 3]
  6.  [Prin][+]  <-11->  Sto* [ 3] -> Pr** [11]
  7.  [Admi][+]  <-19->  Sto* [ 3] -> Pr** [11] -> Ad** [19]
  8.  [Secr][+]  <-20->  Sto* [ 3] -> Pr** [11] -> Ad** [19] -> Secr [20]
  9.  [Conf][+]  <-15->  Sto* [ 3] -> Pr** [11] -> Ad** [19] -> Secr [20] -> Co** [15]
 10.  [DEL ][+]  <-20->  Sto* [ 3] -> Pr** [11] -> Ad** [19] -> Secr [20]
 11.  [DEL ][+]  <-19->  Sto* [ 3] -> Pr** [11] -> Ad** [19]
 12.  [DEL ][+]  <-11->  Sto* [ 3] -> Pr** [11]
 13.  [Conf][+]  <-11->  Sto* [ 3] -> Pr** [11] -> Co** [11]
 14.  [WAIT][+]  <-19->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19]
 15.  [CAR ][+]  <-19->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19] -> CAR  [19]
 16.  [Admi][+]  <-27->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19] -> CAR  [19] -> Ad** [27]
 17.  [Secr][+]  <-22->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19] -> CAR  [19] -> Ad** [27] -> Secr [22]
 18.  [DEL ][+]  <-27->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19] -> CAR  [19] -> Ad** [27]
 19.  [DEL ][+]  <-19->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19]
 20.  [DEL ][+]  <-11->  Sto* [ 3] -> Pr** [11]
 21.  [DEL ][+]  <- 3->  Sto* [ 3]
 22.  [DEL ][+]  <- 0->
 23.  [DEL ][-]  <- 0->
 24.  [Admi][+]  <- 8->  Ad** [ 8]
 25.  [CAR ][+]  <- 8->  Ad** [ 8] -> CAR  [ 8]
 26.  [Stor][+]  <-11->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11]
 27.  [Secr][+]  <-12->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Secr [12]
 28.  [Conf][+]  <- 7->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Secr [12] -> Co** [ 7]
 29.  [DEL ][+]  <-12->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Secr [12]
 30.  [DEL ][+]  <-11->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11]
 31.  [Conf][+]  <-11->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11]
 32.  [WAIT][+]  <-19->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19]
 33.  [Prin][+]  <-27->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Pr** [27]
 34.  [Secr][+]  <-22->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Pr** [27] -> Secr [22]
 35.  [DEL ][+]  <-27->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Pr** [27]
 36.  [DEL ][+]  <-19->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19]
 37.  [Secr][+]  <-20->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Secr [20]
 38.  [Prin][+]  <-28->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Secr [20] -> Pr** [28]
 39.  [FIN ][+]  <-28->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Secr [20] -> Pr** [28] -> FIN  [28]

Final solution:
   Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Secr [20] -> Pr** [28] -> FIN  [28]
Optimum solution (number: 1):
   Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Secr [20] -> Pr** [28] -> FIN  [28]
```

Fig. 6.16: Example for the course of planning, performed by EVA on PAD task 4 (printout from a log file generated by the PAD simulation system)

```
================================================================
69. PAD-task (4) loaded for EVA...  (5 Locations) (OptRating: 28)
================================================================

Sto [1]:  10:00 - 12:15  (10)
Sec [0]:  11:00 - 16:00  (10)
Adm [2]:  10:00 - 14:30  (90)
Pri [2]:  10:00 - 16:00  (90)
Con [2]:  13:00 - 13:00  (60)

        Post   Café   Off    Stor   Sec    Adm    Pri    Conf   Cen    CAR    WAIT   DEL    FIN
  1.  | 0.00 | 0.00 | 0.00 >>0.68<< 0.39 | 0.57 | 0.13 | 0.28 | 0.00 | 0.58 | 0.00 | 0.00 | 0.02 |  [+]  MemMax:  1 (0.00)  <- 3->  Sto* [ 3]
  2.  | 0.00 | 0.00 | 0.00 | 0.00 | 0.25 | 0.28 >>0.31<< 0.26 | 0.00 | 0.28 | 0.00 | 0.03 | 0.05 |  [+]  MemMax:  5 (0.05)  <-11->  Sto* [ 3] -> Pr** [11]
  3.  | 0.00 | 0.00 | 0.00 | 0.19 | 0.26 | 0.29 | 0.00 >>0.96<< 0.00 | 0.30 | 0.00 | 0.03 | 0.09 |  [+]  MemMax:  1 (0.00)  <-11->  Sto* [ 3] -> Pr** [11] -> Co** [11]
  4.  | 0.00 | 0.00 | 0.00 | 0.15 | 0.20 | 0.19 | 0.15 | 0.00 | 0.00 | 0.09 >>1.00<< 0.00 | 0.11 |  [+]  MemMax:  1 (0.00)  <-19->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19]
  5.  | 0.00 | 0.00 | 0.00 | 0.25 | 0.52 | 0.26 | 0.25 | 0.00 | 0.00 >>0.54<< 0.00 | 0.00 | 0.17 |  [+]  MemMax:  1 (0.00)  <-19->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19] -> CAR  [19]
  6.  | 0.00 | 0.00 | 0.00 | 0.33 >>0.49<< 0.42 | 0.24 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.16 |  [+]  MemMax:  1 (0.00)  <-20->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19] -> CAR  [19] -> Secr [20]
  7.  | 0.00 | 0.00 | 0.00 | 0.26 | 0.00 >>0.75<< 0.24 | 0.28 | 0.00 | 0.00 | 0.00 | 0.00 | 0.21 |  [+]  MemMax:  1 (0.00)  <-15->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19] -> CAR  [19] -> Secr [20] -> Ad** [15]
  8.  | 0.00 | 0.00 | 0.00 | 0.09 | 0.11 | 0.00 | 0.08 | 0.11 | 0.00 | 0.00 | 0.00 >>0.80<< 0.36 |  [+]  MemMax:  1 (0.00)  <-20->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19] -> CAR  [19] -> Secr [20]
  9.  | 0.00 | 0.00 | 0.00 | 0.26 | 0.00 | 0.01 | 0.22 | 0.27 | 0.00 | 0.00 | 0.40 >>0.59<< 0.47 |  [+]  MemMax:  6 (1.00)  <-19->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19]
 10.  | 0.01 | 0.00 | 0.00 | 0.24 >>0.46<< 0.21 | 0.29 | 0.00 | 0.00 | 0.00 | 0.00 | 0.37 | 0.18 |  [+]  MemMax: 10 (1.00)  <-20->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19] -> Secr [20]
 11.  | 0.00 | 0.00 | 0.04 | 0.24 | 0.00 | 0.23 >>0.28<< 0.24 | 0.00 | 0.10 | 0.02 | 0.00 | 0.26 |  [+]  MemMax:  4 (0.04)  <-15->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19] -> Secr [20] -> Pr** [15]
 12.  | 0.00 | 0.00 | 0.00 | 0.15 | 0.14 | 0.12 | 0.00 | 0.21 | 0.00 | 0.12 | 0.38 >>0.80<< 0.19 |  [+]  MemMax:  1 (0.00)  <-20->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19] -> Secr [20]
 13.  | 0.00 | 0.00 | 0.01 | 0.29 | 0.00 | 0.24 | 0.02 | 0.34 | 0.00 | 0.45 | 0.18 >>0.45<< 0.32 |  [+]  MemMax:  7 (1.00)  <-19->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19]
 14.  | 0.01 | 0.00 | 0.00 | 0.22 | 0.05 | 0.25 >>0.31<< 0.00 | 0.00 | 0.00 | 0.01 | 0.30 | 0.16 |  [+]  MemMax:  5 (1.00)  <-14->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19] -> Pr** [14]
 15.  | 0.00 | 0.00 | 0.00 | 0.13 | 0.25 | 0.10 | 0.00 | 0.16 | 0.00 | 0.13 | 0.08 >>0.80<< 0.20 |  [+]  MemMax:  1 (0.00)  <-19->  Sto* [ 3] -> Pr** [11] -> Co** [11] -> WAIT [19]
 16.  | 0.01 | 0.00 | 0.00 | 0.24 | 0.08 | 0.17 | 0.01 | 0.00 | 0.00 | 0.11 | 0.00 >>0.75<< 0.24 |  [+]  MemMax:  7 (1.00)  <-11->  Sto* [ 3] -> Pr** [11]
 17.  | 0.00 | 0.00 | 0.00 | 0.19 | 0.17 | 0.25 | 0.00 | 0.04 | 0.01 | 0.24 | 0.00 >>0.63<< 0.18 |  [+]  MemMax:  8 (1.00)  <- 3->  Sto* [ 3]
 18.  | 0.01 | 0.00 | 0.00 | 0.00 | 0.28 | 0.29 | 0.02 | 0.31 | 0.00 | 0.27 | 0.00 >>0.38<< 0.14 |  [+]  MemMax:  7 (1.00)  <- 0->
 19.  | 0.01 | 0.00 | 0.00 | 0.02 | 0.28 >>0.30<< 0.20 | 0.24 | 0.00 | 0.25 | 0.00 | 0.11 | 0.10 |  [+]  MemMax:  4 (1.00)  <- 8->  Ad** [ 8]
 20.  | 0.00 | 0.00 | 0.00 | 0.37 | 0.37 | 0.00 | 0.29 | 0.26 | 0.00 >>0.74<< 0.00 | 0.03 | 0.15 |  [+]  MemMax:  1 (0.00)  <- 8->  Ad** [ 8] -> CAR  [ 8]
 21.  | 0.00 | 0.00 | 0.00 >>0.71<< 0.28 | 0.00 | 0.30 | 0.69 | 0.00 | 0.00 | 0.00 | 0.11 | 0.18 |  [+]  MemMax:  1 (0.00)  <-11->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11]
 22.  | 0.00 | 0.00 | 0.00 | 0.00 | 0.27 | 0.18 | 0.43 >>0.86<< 0.00 | 0.00 | 0.02 | 0.04 | 0.19 |  [+]  MemMax:  1 (0.00)  <-11->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11]
 23.  | 0.00 | 0.00 | 0.00 | 0.11 | 0.18 | 0.11 | 0.22 | 0.00 | 0.00 | 0.00 >>1.00<< 0.01 | 0.18 |  [+]  MemMax:  1 (0.00)  <-19->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19]
 24.  | 0.00 | 0.00 | 0.00 | 0.20 | 0.61 | 0.23 >>0.92<< 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.33 |  [+]  MemMax:  1 (0.00)  <-27->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Pr** [27]
 25.  | 0.00 | 0.00 | 0.00 | 0.34 >>0.45<< 0.25 | 0.00 | 0.38 | 0.00 | 0.00 | 0.00 | 0.00 | 0.25 |  [+]  MemMax:  1 (0.00)  <-22->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Pr** [27] -> Secr [22]
 26.  | 0.00 | 0.00 | 0.00 | 0.06 | 0.00 | 0.02 | 0.03 | 0.04 | 0.00 | 0.00 | 0.00 >>0.80<< 0.20 |  [+]  MemMax:  1 (0.00)  <-27->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Pr** [27]
 27.  | 0.00 | 0.00 | 0.01 | 0.27 | 0.02 | 0.29 | 0.00 | 0.36 | 0.00 | 0.00 | 0.00 >>0.66<< 0.49 |  [+]  MemMax:  5 (1.00)  <-19->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19]
 28.  | 0.00 | 0.00 | 0.00 | 0.21 | 0.49 | 0.18 | 0.04 | 0.00 | 0.00 | 0.00 | 0.00 >>0.52<< 0.26 |  [+]  MemMax:  7 (1.00)  <-11->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11]
 29.  | 0.00 | 0.00 | 0.00 | 0.00 | 0.32 | 0.18 >>0.37<< 0.04 | 0.00 | 0.00 | 0.00 | 0.21 | 0.14 |  [+]  MemMax:  8 (1.00)  <-19->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Pr** [19]
 30.  | 0.00 | 0.00 | 0.00 | 0.24 >>0.45<< 0.22 | 0.00 | 0.31 | 0.00 | 0.00 | 0.00 | 0.00 | 0.19 |  [+]  MemMax:  4 (0.02)  <-20->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Pr** [19] -> Secr [20]
 31.  | 0.00 | 0.00 | 0.00 | 0.31 | 0.00 | 0.24 | 0.26 >>0.35<< 0.00 | 0.00 | 0.02 | 0.00 | 0.21 |  [+]  MemMax:  1 (0.00)  <-15->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Pr** [19] -> Secr [20] -> Co** [15]
 32.  | 0.00 | 0.00 | 0.01 | 0.13 | 0.09 | 0.11 | 0.12 | 0.00 | 0.00 | 0.00 | 0.00 >>0.80<< 0.14 |  [+]  MemMax:  9 (0.03)  <-20->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Pr** [19] -> Secr [20]
 33.  | 0.00 | 0.00 | 0.00 | 0.36 | 0.00 | 0.24 | 0.21 | 0.00 | 0.00 | 0.00 | 0.02 >>0.68<< 0.20 |  [+]  MemMax:  8 (1.00)  <-19->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Pr** [19]
 34.  | 0.00 | 0.00 | 0.00 | 0.22 | 0.02 | 0.19 | 0.00 | 0.24 | 0.00 | 0.00 | 0.00 >>0.47<< 0.20 |  [+]  MemMax:  5 (1.00)  <-11->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11]
 35.  | 0.00 | 0.00 | 0.00 | 0.00 >>0.29<< 0.20 | 0.03 | 0.15 | 0.00 | 0.00 | 0.00 | 0.22 | 0.16 |  [+]  MemMax:  7 (1.00)  <-12->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Secr [12]
 36.  | 0.00 | 0.00 | 0.00 | 0.20 | 0.00 | 0.19 >>0.58<< 0.31 | 0.00 | 0.00 | 0.00 | 0.08 | 0.24 |  [+]  MemMax:  3 (0.02)  <-20->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Secr [12] -> Pr** [20]
 37.  | 0.00 | 0.00 | 0.01 | 0.20 | 0.27 | 0.22 | 0.00 >>0.28<< 0.00 | 0.00 | 0.00 | 0.00 | 0.17 |  [+]  MemMax:  1 (0.00)  <-15->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Secr [12] -> Pr** [20] -> Co** [15]
 38.  | 0.00 | 0.00 | 0.01 | 0.16 | 0.11 | 0.11 | 0.11 | 0.00 | 0.00 | 0.00 | 0.00 >>0.80<< 0.21 |  [+]  MemMax:  8 (0.00)  <-20->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Secr [12] -> Pr** [20]
 39.  | 0.00 | 0.00 | 0.00 | 0.32 | 0.28 | 0.25 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 >>0.66<< 0.20 |  [+]  MemMax:  8 (1.00)  <-12->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Secr [12]
 40.  | 0.00 | 0.00 | 0.00 | 0.22 | 0.00 | 0.17 | 0.02 | 0.32 | 0.00 | 0.00 | 0.00 >>0.48<< 0.20 |  [+]  MemMax:  7 (1.00)  <-11->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11]
 41.  | 0.00 | 0.00 | 0.00 | 0.00 | 0.05 | 0.22 | 0.07 >>0.32<< 0.00 | 0.00 | 0.00 | 0.17 | 0.16 |  [+]  MemMax:  5 (1.00)  <-11->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11]
 42.  | 0.00 | 0.00 | 0.00 | 0.12 | 0.17 | 0.11 | 0.24 | 0.00 | 0.00 | 0.00 >>1.00<< 0.00 | 0.17 |  [+]  MemMax:  1 (0.00)  <-19->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19]
 43.  | 0.00 | 0.00 | 0.00 | 0.19 >>0.43<< 0.21 | 0.07 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.28 |  [+]  MemMax:  7 (1.00)  <-20->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Secr [20]
 44.  | 0.00 | 0.00 | 0.00 | 0.33 | 0.00 | 0.25 >>0.89<< 0.33 | 0.00 | 0.00 | 0.00 | 0.00 | 0.54 |  [+]  MemMax:  1 (0.00)  <-28->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Secr [20] -> Pr** [28]
 45.  | 0.00 | 0.00 | 0.00 | 0.43 | 0.36 | 0.28 | 0.00 | 0.37 | 0.00 | 0.00 | 0.00 | 0.00 >>0.69<<  [+]  MemMax:  1 (0.00)  <-28->  Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Secr [20] -> Pr** [28]
                                                                                                                     -> FIN  [28]

Final solution:
   Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Secr [20] -> Pr** [28] -> FIN  [28]
Optimum solution (number: 1):
   Ad** [ 8] -> CAR  [ 8] -> Sto* [11] -> Co** [11] -> WAIT [19] -> Secr [20] -> Pr** [28] -> FIN  [28]
```

must confess that the chosen examples are in so far not completely typical, as they both finally lead finally to the optimal plan.)

The courses of planning presented here refer to PAD task 4. The reader should refer to table 5.1 for the specifications of this PAD task to be able to judge the operator application of the human subjects on the one side, and of EVA on the other side (for the instructions of PAD task 4, see appendix A).

The printout from the log files of the PAD simulation system presented in fig. 6.15 and 6.16 provide much information, some of which is described in the following. Fig. 6.15 shows the course of planning for a human subject. For each operator application, one row is added to the numbered list of temporary plans. In each row, the first column after the numbering gives the applied operator (the names of the Movement actions are made up of the first four letters of their respective goal location). Afterwards, "[+]" or "[-]" indicates, if the operator was a valid user input (according to table 4.3). The number shown in acute brackets represents the current score after operator application (according to the advanced evaluation). Right of the current score the current plan is presented as sequence of actions plus *Finish* (stars in location names indicate the priority of the respective appointment). After each action, the present (advanced) score is given in angular brackets. In following, the numbered list from the first to the last operator application gives an overall impression of the whole process of planning.

Fig. 6.16 contains even more information relevant to the course of planning of EVA. Instead of an abbreviation of the applied operator, the numbered list provides the complete activation pattern of EVA's output units for the respective processing cycle in each row. The column titles allow the assignment of each activation value to its respective evaluator/operator. The output unit with maximum activation is highlighted by inverse acute brackets. In this way, one can determine the operator applied by EVA in the respective processing cycle. On the right of each row, the current plan is displayed in the same way as in fig. 6.15.

In fig. 6.16, one can discover, for example, chains of applied *Delete* operators (e.g., in steps 15 to 18). The activation of the *Delete* evaluator decreases slightly from step to step, until another evaluator gets a higher activation value. Vice versa, at the end of the course of planning, one can observe the increasing activation value of the *Finish* evaluator, until it is high enough to terminate the process of planning.

In comparing the course of planning between this human subject and EVA, many differing details are noticeable, but there are also some interesting similarities. For example, in the first half of both courses of planning, in the course of several steps the current plan starts with the sequence "*Move-to-storehouse* → *Move-to-printing-office*". Furthermore, both the human subject and EVA supplement this fragment temporarily with the sequence "*Move-to-conference-room* → *Wait* → *Drive-by-car*". In the second half of both courses of planning, the beginning of the current plan "*Move-to-administration* → *Drive-by-car* → *Move-to-storehouse*" is retained in spite of several *Delete* chains terminated just before deleting *Move-to-storehouse*.

Furthermore, the beginning of the application of the *Delete* operator is always triggered by the same kind of situation (notice steps 9, 17, 28, and 34 for the human subject and steps 7, 11, 14, 25, 31, and 37 for EVA): Namely, when it was not possible to fulfill the scheduled

appointment at the last visited location, which can be diagnosed by the decreasing standard end score for the last action of the respective current plan. Both the human subject and EVA rely on this situational information to begin a phase of plan revision. I will return to this interesting point in the discussion (section 7.3).

However, as pointed out in the beginning, even if this comparison reveals interesting results, they only have a demonstrative character. Therefore, I will return in the following subsections to quantitative comparisons.

6.1.6.2 Relationship between the Length of the Operator Sequence and the End Score

In examining the course of planning both for the human subjects and the simulated subjects, the impression arose, that shorter operator sequences often yield better end scores. If this relationship is really the same for the human sample, as it is for EVA, this would provide an interesting insight into important characteristics of the course of planning. To check this assumption, the correlation coefficients between `Length` and `EndScoreStd` were calculated. They are given in table 6.5.

As the table shows, the correlations are in fact negative. For PAD task 4, EVA yields a more negative coefficient than the human sample (r^{EVA} = -.31 vs. r^{HuSbj} = -.22), for PAD task 5, it is the other way round (r^{EVA} = -.10 vs. r^{HuSbj} = -.21). However, at least the sign of the correlation coefficients and the order of their magnitude is met by EVA. This shows that EVA is even able to reproduce subtle tendencies within the planning process, although they were not considered at all during model fitting.

6.1.6.3 First Used Operator

A deeper analysis of the planning process can also be carried out through determining the operators used in certain states of the PAD world. Depending on the number of considered states and the procedure used for determining these relevant states, such an analysis can reach such a degree of high complexity, that meaningful evaluations and interpretations are nearly impossible. Therefore, in the present study, only two states were taken into consideration,

Table 6.5: Correlations between the length of the operator sequence and the standard end score

	PAD task			
	Nº 4		Nº 5	
	Sample		Sample	
	Humans	EVA	Humans	EVA
Correlation between `Length` and `EndScoreStd`	-.22	-.31**	-.21	-.10**

** These correlations are significant on the level of 0.1%. This is not surprising since n = 4500 for the sample generated by EVA.

which can be defined in a very straightforward manner: First, the starting point of the planning process, and second, the starting point of the finally generated plan. Thus, I examined the use of the first operator in the overall course of planning, and the use of the first action (operator) in the finally generated plan.

Fig. 6.17 shows both kinds of operator use for PAD task 4 in comparison between EVA and the human subjects. For the first operator used in the overall course of planning (white parts of each bar, pure white for the human sample and patterned white for EVA), this comparison reveals quite a different frequency distribution. Even if the three operators with highest frequency are equal for the human sample and EVA (*Move-to-storehouse*, *Move-to-administration*, *Drive-by-car*), their respective frequencies are very different. EVA generates a strong peak for *Move-to-storehouse* (81%), while the human subjects do not show a clear preference (*Move-to-storehouse*: 29%; *Move-to-administration*: 29%; *Drive-by-car*: 13%).

An inspection of the use of the first action in the final plan (black parts of each bar, pure black for the human sample and patterned black for EVA) reveals, that the positions of the three maxima remain unchanged, and also, that their respective frequencies have become more equal. *Move-to-storehouse* is used as first action by the human subjects in 47% of the plans, and by EVA in 32% of the plans. For *Move-to-administration* these percentages are 44% and 37%, respectively. In summary, EVA does not reproduce the use of the first operator in the overall

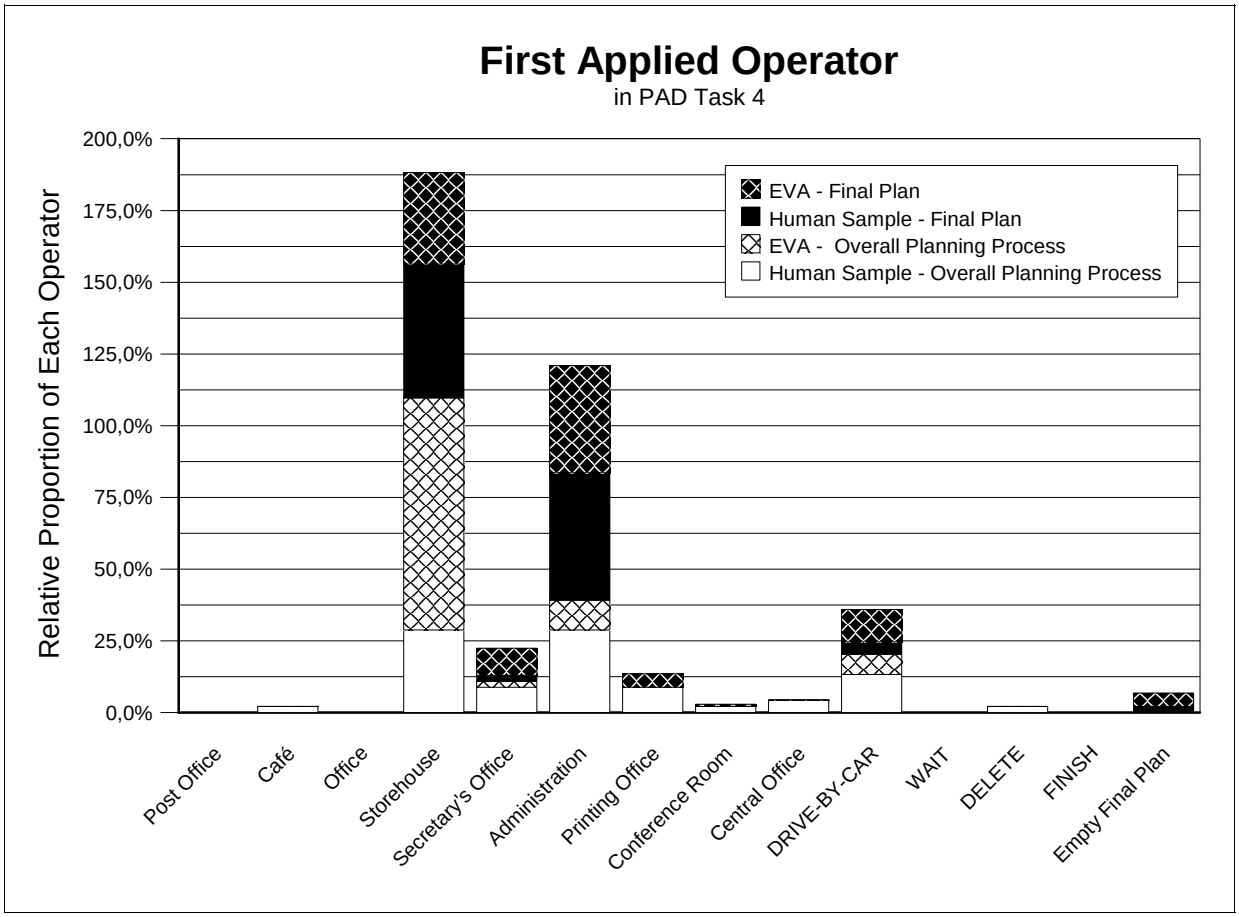

Fig. 6.17: Frequency distribution of applied operators for the starting point of PAD task 4 and for the beginning of the final generated plan

planning process very well, however, the frequency distributions of the first used action in the final plan are quite similar for the human sample and EVA (although model fitting was not directed at all towards these frequency distributions).

The same comparisons can be carried out for PAD task 5 (fig. 6.18). Unfortunately, as an inspection of fig. 6.18 reveals, there are considerable differences between the human sample and EVA. For the first applied operator of the overall planning process, the maxima of frequencies are quite different (for the human sample: *Move-to-administration*: 40%, *Move-to-central-office*: 37.8%; for EVA: *Move-to-conference-room*: 49%, *Wait*: 24%). For the first action of the final plan, these differences become smaller, but are still of considerable size. The human subjects begin the final plan most frequently with *Move-to-printing-office* (56%), while EVA prefers still *Move-to-conference-room* (39%). This different behavior explains partly, why the simulated subjects reach the optimal plan very rarely in comparison with the human subjects. The optimal plan for PAD task 5 begins with *Move-to-printing-office* (see table 5.2), and since EVA "dislikes" to apply this action at the beginning, the simulated subjects are only seldom in the position to reach the optimal plan (for a further discussion see section 7.1).

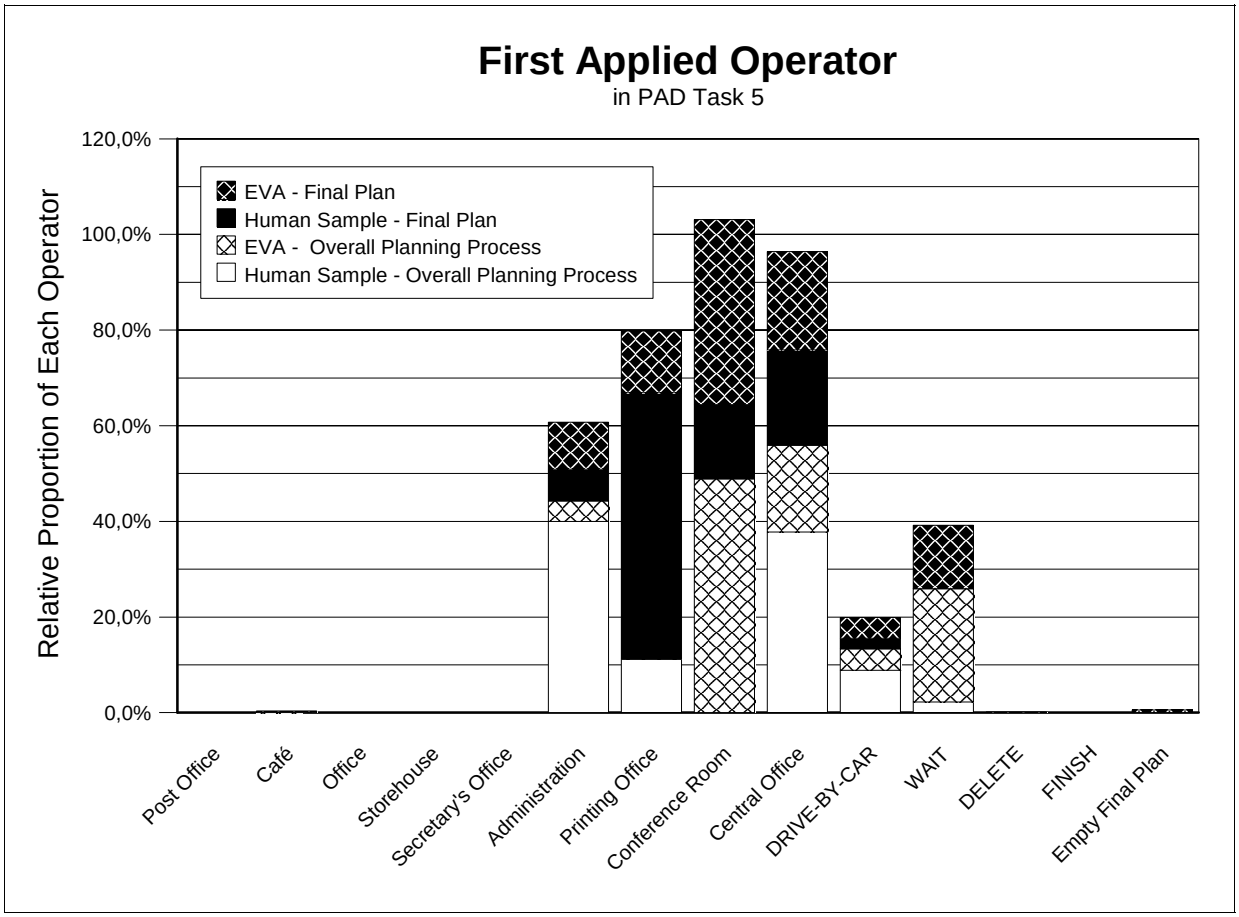

Fig. 6.18: Frequency distribution of applied operators for the starting point of PAD task 5 and for the beginning of the final generated plan

6.1.7 Summary

In summary, the results of model fitting are satisfactory. For PAD task 4, many of the relevant characteristics of the planning behavior of the human sample are met by EVA. Only for `EndScoreStd` (6.1.1), `TimeTooLate`, `TimeTooEarly` (6.1.5), and the application of the first operator in the overall planning process (6.1.6.3) the differences are of notable size.

For PAD task 5, the overall picture is more mixed. Many of the variables used for comparison with the human sample are in the same order of magnitude for both the human and the simulated subjects, but there are also some considerable differences. These differences especially affect the variables `ESOptRatio` and `MSOptRatio` (6.1.1), the frequency distributions for the variables `EndScoreStd` (6.1.1), `Length`, and `NumTotalDel` (6.1.2), as well as the application of the "considering urgency" heuristic (6.1.4), the variable `TimeTooLate` (6.1.5), and the application of the first operator both in the overall planning process and in the final plan (6.1.6.3).

6.2 Performance of EVA on Randomly Generated PAD Tasks

Beginning with this section, the results of applying EVA to new PAD tasks are reported. However, first one must decide which of the two parameter configurations, which result from the second stage of model fitting, one wants to use. In the following, a compromise is chosen. First, a parameter configuration is determined that results from calculating the mean values between the parameter settings for PAD task 4 and PAD task 5. This approach is based on the assumption, that both PAD tasks 4 and 5 have their own special properties that are not covered by the basic specifications of EVA. Thus, mean values as parameter values may be a good choice to optimize the performance of EVA for a broad variety of different PAD tasks. The respective parameter configuration is called "medium configuration" in the following. The parameter values are: $f_D = 0.8$, $f_F = 0.6$, $\mathsf{NoiseProportion} = 0.2$, $\mathsf{MemForgetFactor}_{\mathsf{Simul}} = 0.8$.

Second, for comparison another parameter configuration is used, namely the configuration for PAD task 4. In the following it is called "P4 configuration"; it comprises the known parameter values $f_D = 0.8$, $f_F = 0.75$, $\mathsf{NoiseProportion} = 0.1$, and $\mathsf{MemForgetFactor}_{\mathsf{Simul}} = 0.8$. The reason for employing this configuration in addition to the others is based on the following considerations: Since the goodness of fit is especially good for PAD task 4 and only moderate for PAD task 5 (although for both special parameter settings were chosen), the latter PAD task (and only this one) may have special properties which are not covered by the basic architecture of EVA (including the structure of the training data). Such properties are considered rare. Thus, for PAD tasks in general, it may be an advantage for EVA, if it can work with the parameter settings for PAD task 4.

Since the medium configuration and the P4 configuration are based on contrary assumptions, the comparisons in this and in the next section also serve as clues for determining which of the two assumptions is more plausible.

In this section, EVA is applied to randomly generated PAD tasks of different complexity. The PAD tasks are generated according to the same basic principles and parameter settings as the PAD tasks used for the training data (see section 4.5.2.1 and table 4.5). Only MinSchedulApps and MaxSchedulApps are varied. They are set to an equal value called SchedulApps (this parameter corresponds to the number of scheduled appointments in each PAD task), which is varied to regulate the complexity of the generated PAD tasks. The variables EndScoreStd%, MaxScoreStd%, ESOptRatio, and MSOptRatio are used for assessing EVA's performance. The former two variables correspond closely to the variables of section 6.1.1 with similar names. The additional "%" in their names indicates, that they are given as percentages in relation to the average optimal score for the generated PAD tasks. For table 6.6, ESOptRatio and MSOptRatio (see section 6.1.1 for their meaning) are calculated according to the standard evaluation. In addition, table 6.6 provides the variable Length (number of applied operators in the course of planning).

Table 6.6 reveals the following results: First, the performance indicators are smaller for the medium configuration than for the P4 configuration. This result provides evidence for the assumptions underlying the choice of the P4 configuration. Since both the PAD tasks underlying EVA's training data[25] and the PAD tasks underlying the results of table 6.6 were generated randomly according to the same basic parameter settings, both sets of PAD tasks belong to the same "population". Therefore, since the P4 configuration yields the better performance, PAD task 4 must meet more of the basic characteristics of this population than PAD task 5 does.

Table 6.6: Performance of EVA on randomly generated PAD tasks. The meaning of the variables is explained in the text. n corresponds to the number of randomly generated PAD tasks for each sample.

	SchedulApps (Number of scheduled appointments in each PAD task)						
	3 (n=450)	4 (n=450)	5 (n=450)	6 (n=450)	7 (n=450)	8 (n=450)	9 (n=300)
Medium configuration							
EndScoreStd% [%]	64.0	67.0	70.4	73.1	75.0	73.9	70.4
MaxScoreStd% [%]	92.4	89.0	88.0	88.8	87.7	86.7	85.7
ESOptRatio [%]	41.6	33.6	21.3	15.1	12.7	8.2	4.0
MSOptRatio [%]	68.4	51.8	31.8	23.8	17.8	13.8	8.0
Length	21.6	29.6	35.1	44.9	48.9	63.1	67.6
P4 configuration							
EndScoreStd% [%]	72.5	75.5	78.3	83.0	80.3	79.7	78.5
MaxScoreStd% [%]	93.5	91.3	89.1	90.0	88.0	87.0	87.0
ESOptRatio [%]	50.7	43.1	26.9	21.8	12.2	10.9	9.3
MSOptRatio [%]	72.0	57.6	35.8	26.2	18.4	13.8	13.0
Length	17.5	23.2	26.9	26.8	37.6	36.1	49.0

[25] For a minority of the PAD tasks used for EVA's training data the basic parameter settings were slightly changed. For a precise description see section 4.5.2.2.

Thus, most probably, PAD task 5 has some special qualities only rarely met by randomly generated PAD tasks, even if this random generation is based on overt characteristics of the predefined PAD tasks (see section 4.5.2.1).

Furthermore, the comparison of the medium and the P4 configuration shows, that the operator sequences for the former one are longer than for the latter one. This can be easily explained by the smaller value of f_F for the medium configuration ($f_F = 0.6$) than for the P4 configuration ($f_F = 0.75$). In general, `Length` increases with an increasing number of scheduled appointments. This is a meaningful result, since the need for elaboration grows with enhanced complexity of PAD tasks.

Further, table 6.6 reveals that the best end scores for the P4 configuration (`EndScoreStd%`) are obtained for PAD tasks with six scheduled appointments. This result is in so far surprising, as one may expect the best end scores for PAD tasks with the least number of scheduled appointments. A possible explanation for this phenomenon lies in the fact, that the training data of EVA is based on PAD tasks with at least five scheduled appointments (see the value of MinSchedulApps in table 4.5). However, the variables `MaxScoreStd%`, `ESOptRatio` and `MSOptRatio` obtain their maximum values for PAD tasks with three scheduled appointments (93.5%, 50.7%, and 72.0%, respectively). With an increasing number of scheduled appointments, the values of these variables decrease, until they reach their minimum (when SchedulApps = 9: Then `MaxScoreStd%` = 87.0%, `ESOptRatio` = 9.3%, `MSOptRatio` = 13.0%). The latter two percentages seem to be remarkably high, if one considers the poor performance of EVA for PAD task 5 with regard to `ESOptRatio` and `MSOptRatio`.

To sum up, the results of this section provide further evidence for the general applicability of EVA. Obviously, neither the first nor the second stage of model fitting has led to a model that is specialized on PAD tasks 4 and 5, but which does not have any further applicability. Of course, the approach taken in this section does not claim to be a validation of EVA. This step is reserved for the next section.

6.3 A First Attempt at Validation

The best procedure of validation would be to compare predictions of EVA regarding certain PAD tasks with the results obtained by human samples. However, since EVA is not yet able to reproduce the empirical results for PAD task 5 with a satisfactory precision, it seems exaggerated to start a validation study right from the very beginning. In addition, it is not totally clear which parameter configuration (medium or P4) should be preferred at the present point. For these reasons, a more economical approach was chosen for the estimation of EVA's current prediction qualities. In the study of Funke and Krüger (1995), which was presented in section 3.2, small samples of humans subjects had to work on several predefined PAD tasks. The authors published the values of the variables `EndScoreStd`, `MaxScoreStd`, and `Length` (see table 3.1). Referring to these results, one can realize a limited test of EVA's validity.

Unfortunately, as a comparison of the standard end scores (EndScoreStd) of the human samples between table 3.1 and table 6.1 reveals, the subjects in the study of Funke and Krüger (1995) show worse performance than the subjects used for this study (for PAD task 4: 22.7 to 25.3; for PAD task 5: 20.9 to 22.9). In addition, the number of applied operators (Length) is also different (compare table 3.1 and table 6.2); the subjects in the study of Funke and Krüger needed less than half of the operators the other subjects needed (for PAD task 4: 16.0 to 40.6; for PAD task 5: 17.9 to 43.6). Since EVA was fitted to meet the characteristics of a certain human sample, the question arises: Will EVA be in a position to predict results for other samples with likely different characteristics?

Nevertheless, the comparison between the human samples from the study of Funke and Krüger (1995) and simulated samples generated by EVA in the medium and in the P4 configuration was carried out. From the study of Funke and Krüger two samples were taken: One for predefined PAD tasks 7 and 8 (n=15), and one for predefined PAD tasks 15 and 16 (n=14). These samples were completely assessed in the easiest mode of presentation, which is important since the simulated PAD world is also based on this mode. For EVA, 450 simulation runs were carried out for each PAD task and each configuration.

Table 6.7: Comparison of different samples working on predefined PAD tasks 7, 8, 15, and 16. The data for the human subjects were taken from the study of Funke and Krüger (1995).

	Sample		
	Subjects simulated by EVA (medium configuration) (n=450)	Subjects simulated by EVA (P4 configuration) (n=450)	Human subjects (n=15 for PAD tasks 7 and 8; n=14 for PAD tasks 15 and 16)
PAD task 7			
EndScoreStd% [%]	87.8*	88.3	83.5
MaxScoreStd% [%]	87.9	88.3*	90.6
Length	13.9*	10.3	13.0
PAD task 8			
EndScoreStd% [%]	77.9	80.3*	90.8
MaxScoreStd% [%]	80.1	81.2*	90.8
Length	29.2	17.8*	13.2
PAD task 15			
EndScoreStd% [%]	85.1	84.1*	84.2
MaxScoreStd% [%]	86.0*	84.5	89.3
Length	28.6	15.3*	17.6
PAD task 16			
EndScoreStd% [%]	79.4	83.4*	83.8
MaxScoreStd% [%]	83.2	84.8*	88.2
Length	52.6	32.2*	22.9

* The asterisk indicates that the difference between the respective value and the corresponding value for the human sample is smaller than the difference between the corresponding values for the other configuration of EVA and the human sample.

Table 6.7 shows the results for the variables EndScoreStd, MaxScoreStd, and Length. An asterisk indicates which of EVA's configurations shows a better fit to the human data. Again, the P4 configuration is superior to the medium configuration. For PAD tasks 8, 15, and 16, the fit between the P4 configuration and the human samples is better than the fit for the medium configuration. Length is especially overestimated by EVA in the medium configuration. For the P4 configuration, this overestimation is replaced by an alternating under- and overestimation with smaller deviations.

For PAD tasks 15 and 16, EndScoreStd is predicted by EVA in the P4 configuration with a remarkably high precision. However, for PAD tasks 7 and 8 the differences are considerably larger. MaxScoreStd is mostly underestimated by EVA in the P4 configuration. The largest difference amounts to 9.6% (for PAD task 8). Unfortunately, EVA is not capable of predicting the rank order of different PAD tasks with regard to the different indicator values in any configuration. This is true for EndScoreStd, for MaxScoreStd, and for Length. Only partial rank orders can be predicted.

The results of this approach to validation are as mediocre as the goodness of its preconditions. Nevertheless, at least for the prediction of the end score and for a rough estimation of the number of applied operators the P4 configuration EVA seems to have certain beneficial qualities. Furthermore, these results yield even more evidence for the claim, that the P4 configuration is better suited to meet a broad variety of PAD tasks than the medium configuration. This confirms the assumption that PAD task 5 has some unusual characteristics in its configuration of scheduled appointments. This topic is discussed more deeply in section 7.1.

7 Discussion

7.1 The Failure of EVA on PAD Task 5

Before starting a deeper discussion, the failure of EVA on PAD task 5 must be considered. One of the remarkable differences between the human sample and the simulated sample refers to the indicators `ESOptRatio` and `MSOptRatio` (table 6.1). While more than 37% of the human subjects reach the optimal end score, the same is true for only 6.3% of the simulated subjects. Why does EVA fail to find the optimal plan with a comparable frequency? A first explanation was given in sections 6.2 and 6.3 with regard to the comparison of the medium and the P4 configuration. Obviously, some of the characteristics of PAD task 5 are not typical for the randomly generated PAD tasks in the training data. In the following, I will try to assess the crucial features of PAD task 5.

Table 5.2 shows, that the optimal plan for PAD task 5 begins with a movement to the printing office. The frequency distribution for the first applied operators (fig. 6.18) reveals, that at least 11% of the human subjects begin their overall planning process with *Move-to-printing-office*. For their final plan this tendency is even reinforced: More than 55% of the subjects start their final plan with this action. On the contrary, at the beginning of the overall planning process, EVA ignores the printing office completely and the final plan is begun by EVA only with a proportion of 13% with *Move-to-printing-office*. Instead, EVA "prefers" to move first to the conference room.

Obviously, the appointment at the printing office lacks some characteristics that would enhance its attraction for EVA. First, its priority is only normal (and the lowest of all scheduled appointments). Second, in regards to the earliest task starting time, it has two competitors with the same value (10 a.m.). Third, its urgency is not especially high (the latest task starting time is just at 3 p.m.). When one assumes that EVA relies on such overt characteristics of appointments for their evaluation, then the prospects of the printing office to be visited first are not very good. The other way round, the fact that EVA fails in PAD task 5, where a deeper assessment of the task configuration is obviously necessary, demonstrates that the evaluation carried out by EVA is most likely restricted to overt characteristics of the current PAD task and current state. Later on, we will see if this restriction is desirable or not. Most likely, is actually rare to find such a coincidence of unfavorable features for an important appointment in randomly generated PAD tasks, and thus, EVA had no adequate opportunity to learn to handle such task configurations. In section 7.3, this technical explanation for EVA's failure in PAD task 5 is supplemented with more psychological considerations.

Since such a central appointment like the one at the printing office is evaluated differently by the human and the simulated subjects, the other differences between these samples with regard to PAD task 5 are not surprising. Therefore, a further analysis has been omitted.

Nevertheless, the different values for NoiseProportion and f_F for fitting PAD tasks 4 and 5 have to be considered. Most likely, the higher value of NoiseProportion for PAD task 5 (0.3 vs.

0.1 for PAD task 4) serves to give EVA a slight push towards the printing office. For smaller values of NoiseProportion, the values of `ESOptRatio` and `MSOptRatio` are decreasing, while the end and max scores are increasing (due to the higher precision because of less noise).

The small value for f_F for fitting EVA to PAD task 5 ($f_F = 0.42$) is caused by the failure of EVA-c to recognize that unfulfilled appointments still are present. This failure arises specifically for PAD task 5. Even when only five of the six scheduled appointments are fulfilled (with an end score of 23; the optimal score is 26), the output unit of EVA-c is maximally activated. Therefore, EVA produces very short courses of planning for PAD task 5 (too short in comparison with the human sample), when f_F is in the same order of magnitude as for PAD task 4 ($f_F = 0.75$).

7.2 Initial Claims of EVA – a Review

EVA is a connectionist model for the simulation of human planning behavior with regard to Plan-A-Day (PAD). PAD is a diagnostic instrument for the assessment of planning capabilities, which, at its current stage of development, is at least usable for research (see section 3.2). PAD shares characteristics of problem solving in general as well as characteristics of planning in a more narrow sense (see section 3.3).

First of all, EVA is not a theory for the explanation of human planning behavior in the PAD world. On the contrary, one of the main goals in the development of EVA was to use as few theories on the functional level as possible. A possible procedure of model development would have been to analyze the planning process of human subjects in advance and to extract as many basic principles as possible. Then, one would have been able to implement these principles by several specialized sub-networks. However, this would have been nothing more than a connectionist implementation of a functional theory derived from specific empirical data.

A related approach would be to develop a more general theory of planning or problem solving in advance (or to use an already established theory) and to apply this theory to the PAD world. Afterwards, one could implement basic functional sub-modules of this theory through connectionist networks. A similar approach is proposed by McCloskey (1991), who criticizes the lack of theoretical foundations in many connectionist models within psychology.

Despite this criticism, EVA is actually conceptualized as a kind of black box, for which nobody knows exactly what is happening inside. However, since EVA does not claim to represent a theory of human planning behavior, this lack of insight is not grave. Moreover, nobody would seriously consider that human beings have the networks EVA-a, EVA-b, and EVA-c in their brains which are activated whenever a PAD task arises in the environment.

Nevertheless, even if the inner structure of EVA's sub-networks lacks any functional theory, EVA, considered as a whole, makes a strong theoretical point. It was along these theoretical lines that EVA was developed. EVA is a pattern recognition and pattern transformation device, which shares many of its basic properties with the networks in the human brain. However, are these natural neural networks are capable of more than pattern classification, pattern

recognition, pattern transformation, etc.? On the current level of knowledge in neuroscience, the answer to this question is most likely no (Clark, 1997). How can pattern recognition devices like our brains perform tasks like problem solving or planning? Some theorists (Clark, 1997; Suchman, 1987; Rumelhart, Smolensky, et al., 1986) explain these capabilities by situated cognitions or actions, as I already pointed out in section 4.1. According to this approach, only a well-structured environment can lead to the next adequate operator application, and many operator applications are directed towards creating and maintaining this external structure. In a broader sense, the "environment" can also comprise internal entities which are currently present as input for the basic pattern recognition devices. For example, mental models belong to this class of inner structures. By using such mental models, e.g., a chess player can consider many future moves in advance.

However, most basically, when no mental model is available, subjects rely heavily on the current structure of the environment to determine their next "move", claims the approach of situated cognition/action. In this framework, EVA is intended to demonstrate that a simple connectionist network, which is only able to perform simple pattern transformations, is capable of simulating the planning process shown by human subjects. If EVA succeeds, this would be even more impressive, since the number of EVA's processing units is very, very small in comparison with the number of processing units in the human brain, and since EVA was not trained to imitate human planning behavior, but instead to recognize the shortest way to the optimal plan. Especially the last point is important. Multi-layer networks are applicable to a broad variety of problems. With an appropriate number of units in their hidden layers, they are capable of approximating nearly every multivariate function (Hornik et al., 1989). Thus, both the explanatory and demonstrative value of a multi-layer network trained to mimic human behavior is comparatively low. Later correspondences between the behavior of the network and the behavior of human beings are noteworthy only if the training relies on principles independent from overt human behavior. For this reason, two of the characteristics of EVA are not surprising at all: First, that EVA generates chains of *Delete* operators, and second, that EVA terminates the planning process on its own sometimes. Both behaviors were observed in the human subjects and afterwards explicitly taken into the training data for EVA-b and EVA-c.

In the beginning I stated that for the development of EVA as little theory as possible was used. By such a strategy, the few theoretical claims that are made get even more support in the event of success. In the following, I will reconsider EVA's basic specifications with regard to their underlying assumptions.

- The use of multi-layer networks is mostly based on technical considerations, but also on the objective to choose a type of neural network with a minimum of biological plausibility (see section 4.1).
- The use of recurrent connections for EVA-b and EVA-c is only necessary to mimic the application of *Delete* chains, as well as assumed motivational influences on the use of the *Finish* operator.

- The subdivision of EVA into sub-networks is mostly due to technical reasons. However, such a subdivision may also be plausible with regard to the different types of cognitive functions realized by each network (action selection [EVA-a] vs. plan revision [EVA-b] vs. meta-cognitive reasoning [EVA-c]).
- The input representation is specified with the objective to resemble the input human subjects receive from the computer screen. This corresponds to the approach of situated action. In addition, inner input is provided by the external situation-action memory. Such inner input is not inconsistent with the notion of situated action.
- The training data is constructed due to the basic principle of evaluating each operator with regard to the best possible end score in the event of its application. Therefore, if the training had been completely successful, EVA would always find the shortest way to the optimal plan. The psychological idea behind this procedure is the assumption, that human subjects are also "trained" through daily experience to find the most suitable solutions to everyday planning problems. Even if the "training success" varies from person to person, it is assumed, that the "error correction" works into the direction of optimal plans, as it is implemented in EVA's training data.
- For EVA-b and EVA-c, the special changes in the basic training data were made for imitating the application of *Delete* chains as well as assumed motivational influences on the use of the *Finish* operator.
- Furthermore, in the first stage of model fitting these basic specifications were adapted to improve the fit between the human sample and the sample simulated by EVA.
- In the second stage of model fitting, the values for some adjustable model parameters were determined. These adjustments were carried out with regard to the goodness of fit for eight indicators describing the planning process.

In summary, with regard to theory, EVA is based mainly on the idea, that a simple neural network, which receives the current state of its environment as input, is capable of simulating human planning behavior in a restricted domain by pure pattern transformation. The additional assumptions and considerations mostly serve the solution of technical problems or the specification of concrete details of the implementation, but nevertheless, some of them are also of psychological interest.

7.3 Assessing the Results

In fitting EVA to empirical data, the results were fairly mixed. The resulting overall fit for PAD task 4 is remarkably good, while certain differences remain for PAD task 5. With regard to PAD task 4, not only do the variables that were subject to model fitting show similar values for the human and the simulated sample, but also variables and distributions that were not intentionally fitted at all. Thus, many of the emergent properties of EVA are closely related to the characteristics of the human planning process: These emergent properties are the application

frequency of each operator, the use of heuristics, the proportion of arrivals that were too late or too early, the application frequency of each action for starting the final plan, and several more. The inspection of the individual course of planning in section 6.1.6.1 reveals, that the specific human subject and EVA always started the application of a chain of *Delete* operators in the same kind of configuration of the environment.

Thus, with regard to PAD task 4, EVA really shows, that pure transformation of an input pattern provided by the environment into an evaluation of applicable operators is sufficient to simulate human planning behavior in the PAD world, at least with regard to the assessed characteristics of the planning process. This is a strong argument in favor of the approach of situated action. Of course, one cannot rule out the possibility, that human subjects plan several steps in advance. In the case of PAD task 4, however, this is obviously not necessary for explaining the performance reached by human subjects.

In section 7.1, with regard to PAD task 5, I concluded, that EVA most likely relies on overt characteristics of the current state of the PAD world to determine its evaluations. This is not surprising for a standard multi-layer network the main capability of which is pattern transformation. Also as we saw, in the theoretical frame of situated action, this restriction is not undesirable at all. However, the comparatively poor fit of EVA on PAD task 5 questions the considerations made so far. In section 7.1, a view of the overt causes of EVA's failure was given, which provided a technical explanation. In this section, possible explanations on a psychological background are considered.

First, one may argue, that relying on overt characteristics of the current state and transforming them is actually enough to explain the behavior of the human subjects. Unfortunately, however, the relative weights assigned to different environmental cues (and also the relative weights assigned to cue interactions) are not the same for EVA and for the human subjects. This may be plausible, since the learning history of human subjects is most likely very different from the learning history of EVA. EVA was trained by a sample of PAD tasks taken from the population defined by the parameters in table 4.5. This definition corresponds to the overt characteristics of predefined PAD tasks, which is important to consider when forming the restrictions for the set of training patterns, but also arbitrary in another respect, because most likely none of the assessed human subjects has ever seen a PAD task before. In light of this consideration, it is remarkable, that the overall results obtained by EVA are very close to empirical results obtained by human samples.

Second, it is also possible, that some processes take place in the human mind, that are compatible with the notion of situated action and pattern transformation, but which are not covered by EVA. For example, the relative weights for the different environmental cues may change according to a higher order process. Or the human subjects may use a mental model of the PAD world to perform a kind of inner "simulation" before carrying out certain steps. Moreover, this mental model may be learned in the first stage of working on a PAD task and later used.

Especially for the latter account, standard connectionist models are not suited. The criticism of Barnden and Pollack (1991) (see section 2.3) is justified in this respect: Advanced features

like fast learning and variable binding must be explainable by connectionist models to enhance their explanatory power.

Furthermore, a third possible explanation for EVA's failure in PAD task 5 must be stated. The human sample consists of 45 "brains" creating variance because of their structural and functional differences (on the functional level one may be able to diagnose different "planning styles"). In addition, variance caused by "noise" has to be considered. However, the simulated sample relies on one "brain" (namely, EVA's three sub-networks), and the only source of variance is the noise applied to the input vector. Unfortunately, this shortcoming cannot be recovered directly, since the training of a sample of different instances of EVA would exceed the amount of available calculating time (see section 4.5.4). Therefore, even if the current instance of EVA represents one possible weight configuration approximating the PAD problem, and other instances may show different "behavior", at the moment one has to be content with this one and only instance. In addition, as forerunners of EVA revealed, EVA's "behavior" can be changed due to modifications of the training data. However, when one operates with networks of this size and with such a large training set, and when the training data is held constant, different network instances resulting from different courses of training are quite similar to each other (not with regard to their weight values, but with regard to the resulting indicator values for the planning process).

In light of these considerations, and since many of the results of EVA in PAD task 5 are at least comparable to the results obtained by human subjects, it is still possible to claim that EVA is an impressive demonstration in favor of situated cognition/action. The whole system, comprising several simple multi-layer networks together with a highly structured environment, is able to reproduce important features of the human planning process.

Furthermore, the results presented in section 6.2 with regard to randomly generated PAD tasks show at least, that EVA generates indicator values in the same order of magnitude as human subjects would do, regarding performance and the length of the overall operator sequence. This impression is reinforced by the results of section 6.3, where a first attempt at validation was carried out. Even if this attempt has several weak points which are discussed in that section, the size of the deviations between the simulated and the human samples is quite acceptable. At least for the end score EVA seems to be a useable predictor.

In this respect, it is interesting, that at the current network size of EVA-a, a convergence level regarding the maximum test, has been reached (see section 4.5.4). The result in the maximum test is closely related to the performance of EVA concerning the end score and the proportion of optimal final plans (since EVA-a is the network directly responsible for choosing the most optimal action). Therefore, for a further enlargement of the number of network units or training patterns no substantial improvement with regard to EVA's performance can be expected (at least, when keeping the input and output representation and the structure of the training data constant). This raises the question, whether a fundamental boundary has been reached, which puts a ceiling on the performance of multi-layer networks of the investigated class in relation to PAD-like planning tasks. Moreover, this hypothetical boundary seems to be roughly identical with the level of performance human beings are able to reach (see sections

6.1.1 and 6.3). Even if this is pure speculation, this correspondence can perhaps be explained by the fact, that the neural structures in human brains share at least some crucial properties with the artificial neural structures on which EVA is based. However, based on the present data, such a claim cannot be taken seriously. The first human sample with a remarkably higher performance will send this speculation into the realm of fantasy.

7.4 EVA's Contribution to Cognitive Science

Even Rumelhart, Smolensky, and their colleagues (1986) addressed the question of how sequential thought processes are explainable by "parallel distributed processing". They provided an answer that is closely related to the approach of situated action and situated cognition. They argued that the interplay of manipulating the environment and pattern matching is sufficient to explain high-level cognitive functions (see section 4.1). In addition, they integrated mental models and goals into the PDP framework. Nevertheless, the only concrete model they were able to present was a quite small network capable of playing tic-tac-toe. Therefore, they wrote (Rumelhart, Smolensky, et al., 1986, p. 48): "These ideas are highly speculative and detached from any particular PDP model. They are useful, we believe, because they suggest how PDP models can be made to come into contact with the class of phenomena for which they are, on the face of it, least well suited – that is, essentially sequential and conscious phenomena."

EVA fills this gap. EVA is a PDP model that simulates planning behavior in a way comparable to human beings. Even if EVA is based on a standard model of connectionism, which lacks all of the advanced features Barnden and Pollack (1991) demand for models of high-level cognition, the simulation clearly demonstrates the performance reachable by such a basic network architecture. EVA does the same as Rumelhart, Smolensky, et al. (1986) propose: In each step, EVA transforms the representation of the environment into an action ("pattern matching"), then, by this action the environment is changed ("manipulating"), and afterwards, the altered environment serves as new input.

Thus, EVA is a demonstration that strengthens the claims of two related approaches: First, the basic claims of situated action and situated cognition are confirmed[26], second, the range of application for connectionist models in psychology is extended to the area of high-level cognitive functions like planning or problem solving, which have been mainly reserved for symbol-processing models so far. Now the ideas of Rumelhart, Smolensky, et al. (1986) are no longer detached from any particular PDP model, instead they are realized even by a model from the traditional PDP approach. Furthermore, the domains of problem solving and planning are no

[26] In this thesis, the approaches of situated action and situated cognition are largely identified with each other. Since PAD is placed somewhere between problem solving in general, which corresponds more to situated cognition, and planning in a more narrow sense, which corresponds more to situated action, the missing differentiation between both approaches is not grave. On the contrary, the central position of PAD justifies to extend the conclusions of this thesis equally to both approaches.

longer reserved solely for models from the symbol-processing approach (e.g., production systems).

However, one could go even further in gaining insights from EVA. According to McCloskey (1991), network models should be used like animal models for developing theories corresponding to human systems. He criticizes many forms of connectionist modeling in general, but nevertheless, he writes (McCloskey, 1991, p. 393): "Demonstrations that a network reproduces human phenomena are certainly important, as such demonstrations may contribute to assessing whether a network is similar in relevant respects to the human cognitive mechanism under investigation. However, attention must also be directed toward elucidating the structure and functioning of the network, and applying the insights gained thereby in developing an explicit theory of the human mechanisms."

Regarding the latter points, a closer examination of EVA's sub-networks may reveal additional information of theoretical value. For example, one could begin by analyzing the distribution of weights or temporary activation patterns triggered by specific states. However, in light of the size of EVA's sub-networks the success of such an enterprise is uncertain. Due to the considerable expenditure one has to expect, it would be advisable to improve the fit of EVA on PAD task 5 and perform a serious validation study before further use of this program is undertaken. Afterwards, an analysis as proposed by McCloskey (1991) could be of great theoretical value.

On the present stage of model development, EVA is just a demonstration in favor of the explanatory power of situated actions and the applicability of connectionist models to challenging tasks from the field of problem solving and planning.

References

Anderson, J. R. (1996). *Kognitive Psychologie*. Heidelberg, Berlin, Oxford: Spektrum Akademischer Verlag.

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah (NJ), London (England): Lawrence Erlbaum Associates.

Barnden, J. A., & Pollack, J. B. (1991). Introduction: Problems for high-level connectionism. In J. A. Barnden & J. B. Pollack (eds.), *High-level connectionist models* (pp. 1-16). Norwood (NJ): Ablex Publishing Corporation.

Blankenberger, S. (1992). *Simulation mentaler Vergleiche mit neuronalen Netzwerken.* Braunschweig: University of Braunschweig (doctoral thesis).

Bourbakis, N., & Tascillo, A. (1997). An SPN-neural planning methodology for coordination of two robotic hands with constrained placement. *Journal of Intelligent and Robotic Systems, 19,* 321-337.

Broadbent, D. (1985). A question of levels: Comment on McClelland and Rumelhart. *Journal of Experimental Psychology: General, 114 (2),* 189-192.

Clark, A. (1997). *Being there. Putting brain, body, and world together again.* Cambridge (MA), London (England): A Bradford Book, MIT Press.

Cohen, J. D., Servan-Schreiber, D., & McClelland, J. L. (1992). A parallel distributed processing approach to automaticity. *American Journal of Psychology, 105 (2),* 239-269.

Dörner, D. (1989). *Die Logik des Mißlingens. Strategisches Denken in komplexen Situationen.* Hamburg: Rowohlt.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science, 14,* 179-211.

Elman, J. L., Bates, E. A., Johnson, M. H., Karmiloff-Smith, A., Parisi, D., & Plunkett, K. (1996). *Rethinking innateness. A connectionist perspective on development.* Cambridge (MA): The MIT Press.

Evers, L. (1995). *Planungskompetenz bei Führungskräften.* Bonn: Psychological department of the university of Bonn (unpublished diploma thesis).

Fahlman, S. E., & Lebiere, C. (1990). The cascade-correlation learning architecture. In D. S. Touretzky (ed.), *Advances in neural information processing systems 2* (pp. 524-532). Morgan Kaufmann Publishers.

Feldman, J. A. (1981). A connectionist model of visual memory. In G. E. Hinton & J. A. Anderson (eds.), *Parallel models of associative memory* (pp. 49 – 81). Hillsdale (NJ): Lawrence Erlbaum Associates.

Fiedler, K. (1996). Explaining and simulating judgment biases as an aggregation phenomenon in probabilistic, multiple-cue environments. *Psychological Review, 103 (1),* 193-214.

Fikes, R., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence, 2,* 189-208.

Fodor, J. A. & Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: A critical analysis. In S. Pinker & J. Mehler (eds.), *Connectionism and symbols* (pp. 3-71). Cambridge (MA): MIT Press.

Funke, J., & Fritz, A. (1995). Über Planen, Problemlösen und Handeln. In J. Funke & A. Fritz (eds.), *Neue Konzepte und Instrumente zur Planungsdiagnostik* (pp. 1-45). Bonn: Deutscher Psychologen Verlag.

Funke, J., & Glodowski, A.-S. (1990). Planen und Problemlösen: Überlegungen zur neuropsychologischen Diagnostik von Basiskompetenzen beim Planen. *Zeitschrift für Neuropsychologie, 1 (2),* 139-148.

Funke, J., & Krüger, T. (1995). "Plan-A-Day": Konzeption eines modifizierbaren Instruments zur Führungskräfte-Auswahl sowie erste empirische Befunde. In J. Funke & A. Fritz (eds.), *Neue Konzepte und Instrumente zur Planungsdiagnostik* (pp. 97-120). Bonn: Deutscher Psychologen Verlag.

Greeno, J. G. & Moore, J. L. (1993). Situativity and symbols: Response to Vera and Simon. *Cognitive science, 17,* 49-59.

Hayes-Roth, B., & Hayes-Roth, F. (1979). A cognitive model of planning. *Cognitive Science, 3,* 275-310.

Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory.* New York: Wiley.

Hertzberg, J. (1989). *Planen. Einführung in die Planerstellungsmethoden der Künstlichen Intelligenz.* Mannheim, Vienna, Zurich: BI-Wissenschaftsverlag.

Hertzberg, J. (1995). Planen aus Sicht der künstlichen Intelligenz: Time for a Change. In J. Funke & A. Fritz (eds.), *Neue Konzepte und Instrumente zur Planungsdiagnostik* (pp. 79-96). Bonn: Deutscher Psychologen Verlag.

Hertzberg, J. (1996). Planen. In G. Strube, B. Becker, C. Freksa, U. Hahn, K. Opwis, & G. Palm (eds.), *Wörterbuch der Kognitionswissenschaft* (pp. 501-509). Stuttgart: Klett-Cotta.

Hinton, G. E., & Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In D. E. Rumelhart & J. L. McClelland (eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1: Foundations* (pp. 282-317). Cambridge (MA): MIT Press.

Hinton, G. E., & Shallice, T. (1991). Lesioning an attractor network: Investigations of acquired dyslexia. *Psychological Review, 98 (1),* 74-95.

Hoffman, R. E., Rapaport, J., Ameli, R., McGlashan, H., Harcherik, D., & Servan-Schreiber, D. (1995). A neural network simulation of hallucinated "voices" and associated speech perception impairments in schizophrenic patients. *Journal of Cognitive Neuroscience, 7 (4),* 479-496.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences, 79,* 2554-2558.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayered feed-forward networks are universal approximators. *Neural Networks, 2 (5),* 359-366.

Hussy, W. (1993). *Denken und Problemlösen.* Stuttgart, Berlin, Cologne: Kohlhammer.

Inder, R. (1996). Planning and problem solving. In M. A. Boden (ed.), *Artificial intelligence* (pp. 23-53). San Diego (CA): Academic Press.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks, 1*, 295-307.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation, 3 (1),* 79-87.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics, 43,* 59-69.

Kosko, B. (1987). Bi-directional associative memories. *IEEE Transactions on Systems, Man and Cybernetics, 18 (1),* 49-60.

Kruschke, J. K. (1992). ALCOVE: An exemplar-based connectionist model of category learning. *Psychological Review, 99 (1),* 22-44.

Kruschke, J. K. (1993). Human category learning: Implications for backpropagation models. *Connection Science, 5 (1),* 3-36.

Lewin, K. (1935). *A dynamic theory of personality*. New York: McGraw-Hill.

McClelland, J. L., & Rumelhart, D. E. (1985). Distributed memory and the representation of general and specific information. *Journal of Experimental Psychology: General, 114 (2),* 159-188.

McCloskey, M. (1991). Networks and theories: The place of connectionism in cognitive science. *Psychological Science, 2 (6),* 387-395.

Medin, D. L., & Ross, B. H. (1990). *Cognitive psychology*. Orlando (FL): Harcourt Brace Jovanovich.

Newell, A., & Simon, H. A. (1961). GPS: A program that simulates human thought. In H. Billing (ed.), *Lernende Automaten* (pp. 109-124). Munich: Oldenbourg.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs (NJ): Prentice-Hall.

Norman, D. A. (1993). Cognition in the head and in the world: An introduction to the special issue on situated action. *Cognitive science, 17*, 1-6.

Nosofsky, R. M. (1986). Attention, similarity, and the identification-categorization relationship. *Journal of Experimental Psychology: General, 115 (1),* 39-57.

Opwis, K. (1996). Problemlösen. In G. Strube, B. Becker, C. Freksa, U. Hahn, K. Opwis, G. Palm (eds.), *Wörterbuch der Kognitionswissenschaft* (pp. 520-530). Stuttgart: Klett-Cotta.

Read, S. J., Vanman, E. J., & Miller, L. C. (1997). Connectionism, parallel constraint satisfaction processes, and gestalt principles: (Re)Introducing cognitive dynamics to social psychology. *Personality and Social Psychology Review, 1 (1),* 26-53.

Ribeiro, F., Barthès, J.-P., & Oliveira, E. (1993). Dynamic selection of action sequences. In J.-A. Meyer, H. L. Roitblat, & S. W. Wilson (eds.), *From animals to animats 2. Proceedings of the second international conference on simulation of adaptive behavior* (pp. 189-195). Cambridge (MA), London (England): A Bradford Book, MIT Press.

Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *International conference on neural networks, San Francisco, CA* (pp. 586-591). Piscataway (NJ).

Rojas, Raúl (1996). *Theorie der neuronalen Netze. Eine systematische Einführung.* Berlin, Heidelberg, New York: Springer.

Rumelhart, D. E. (1997). The architecture of mind: A connectionist approach. In J. Haugeland (ed.), *Mind design II. Philosophy, psychology, artificial intelligence* (pp. 205-232). Cambridge (MA): MIT Press.

Rumelhart, D. E., Hinton, G., & Williams, R. (1986). Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland (eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1: Foundations* (pp. 318-362). Cambridge (MA): MIT Press.

Rumelhart, D. E., Smolensky, P., McClelland, J., & Hinton, G. (1986). Schemata and sequential thought processes in PDP models. In D. E. Rumelhart & J. L. McClelland (eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 2: Psychological and biological models* (pp. 7-57). Cambridge (MA): MIT Press.

Schank, R. C., & Abelson, R. P. (1977). *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures.* Hillsdale (NJ): Lawrence Erlbaum Associates.

Schmidhuber, J., & Wahnsiedler, R. (1993). Planning simple trajectories using neural subgoal generators. In J.-A. Meyer, H. L. Roitblat, & S. W. Wilson (eds.), *From animals to animats 2. Proceedings of the second international conference on simulation of adaptive behavior* (pp. 196-202). Cambridge (MA), London (England): A Bradford Book, MIT Press.

Shastri, L. (1991). The relevance of connectionism to AI: A representation and reasoning perspective. In J. A. Barnden & J. B. Pollack (eds.), *High-level connectionist models* (pp. 259-283). Norwood (NJ): Ablex Publishing Corporation.

Smith, E. R. (1996). What do connectionism and social psychology offer each other? *Journal of Personality and Social Psychology, 70 (5),* 893-912.

Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioral and Brain Sciences, 11,* 1-74.

Suchman, A. (1987). *Plans and situated actions.* Cambridge: Cambridge University Press.

Tollenaere, T. (1990). SuperSAB: Fast adaptive backpropagation with good scaling properties. *Neural Networks, 3,* 561-573.

Vera, A. H. & Simon, H. A. (1993). Situated action: A symbolic interpretation. *Cognitive science, 17,* 7-48.

Waloszek, G. (1996). Parallele Gedächtnismodelle. In D. A. Graz & K.-H. Stapf (eds.), *Enzyklopädie der Psychologie. Vol. C-II-4: Gedächtnis* (pp. 261-335). Göttingen, Bern, Toronto, Seattle: Hogrefe, Verlag für Psychologie.

Werbik, H. (1978). *Handlungstheorien.* Stuttgart: Kohlhammer.

Werbos, P. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* Harvard (PhD thesis).

Widrow, G., & Hoff, M. E. (1960). Adaptive switching circuits. In *Institute of radio engineers, western electronic show and convention, convention record, part 4* (pp. 96-104).

Zell, A. (1997). *Simulation neuronaler Netze.* Munich, Vienna: Oldenbourg.

Zimbardo, P. G. (1995). *Psychologie.* Berlin, Heidelberg, New York: Springer.

Appendices

A – Instructions for PAD Tasks 4 and 5

B – Table of Movement Times

C – Data Sheets Generated by the PAD Simulation System

D – Instructions for the Use of NWRun

Appendix A
Instructions for PAD Tasks 4 and 5

Instructions for PAD Task 4

- Between 10.00 a.m. and 0.15 p.m. you are supposed to come to the storehouse and to check the supplies. This will take 10 minutes. IMPORTANT
- Between 11.00 a.m. and 4.00 p.m. you are supposed to come to the secretary and to dictate a letter to her. This will take 10 minutes.
- At the latest at 1.00 p.m. you are supposed to take part in a conference. This conference will go on until 2.00 p.m. VERY IMPORTANT
- By 2.30 p.m. you are supposed to come to the administration and to sign a contract there. For doing this you will need 90 minutes. VERY IMPORTANT
- Between 10.00 a.m. and 4.00 p.m. you are supposed to come to the printing office and to copy a book. This will take 90 minutes. VERY IMPORTANT

Instructions for PAD Task 5

- Between 1.30 p.m. and 2.30 p.m. you can meet a customer at the café. The talk will take 30 minutes. IMPORTANT
- You are supposed to come between 11.00 a.m. and 2.00 p.m. to the office and to deal with files there. For this you will need 60 minutes. VERY IMPORTANT
- At the latest at 11.30 a.m. you are supposed to take part in a conference. This conference will go on until 0.15 p.m. IMPORTANT
- Between 10.00 a.m. and 4.15 p.m. you are supposed to come to your chief to the central office. He wants to meet you for 10 minutes. VERY IMPORTANT
- Between 10.00 a.m. and 4.00 p.m. you are supposed to come to the administration. The work there will take 55 minutes. IMPORTANT
- Between 10.00 a.m. and 3.00 p.m. you are supposed to come to the printing office and to copy a book. This will take 10 minutes.

Appendix B
Table of Movement Times

Movement Times in the PAD World

	Café	Post office	Office	Storehouse	Secretary's office	Administration	Printing office	Conference room	Central office
Café		10	29	30	26	40	55	66	64
Post office	3		19	32	28	30	57	68	54
Office	10	6		40	26	21	64	76	45
Storehouse	10	11	13		14	38	25	36	61
Secretary's office	9	9	9	5		24	39	50	47
Administration	13	10	7	13	8		62	74	24
Printing office	18	19	21	8	13	21		12	86
Conference room	22	23	25	12	17	25	4		97
Central office	21	18	15	20	16	8	29	32	

Movement times with car use [minutes]

Movement times without car use [minutes]

Appendix C
Data Sheets Generated by the PAD Simulation System

Remark: In the following, the term “solution” refers to the term “plan”, and the term “rating” refers to the term “score”.

Human Subjects - PAD Task 4

```
EVALUATION OF PLANNING PROCESSES WITHIN PLAN-A-DAY
==================================================

Number of considered planning processes: 45
Number of optimum solutions            : 45
Average number of OSs for each PAD-task: 1.00

                  Mean        SD          Min         Max
-------------------------------------------------------------
End rating (Std.) 25.29        5.08         0          28    (OS%: 90.32 %)
Max rating (Std.) 26.49        2.09        19          28    (OS%: 94.60 %)
End rat. (Adv.)   22.18        5.65         0          28    (OS%: 79.21 %)
Max rat. (Adv.)   26.36        2.58        15          28    (OS%: 94.13 %)
Opt. Rating       28.00        0.00        28          28

For standard evaluation...
Opt. end rating reached: 10  (in 22.22 % of pp.)
Opt. max rating reached: 11  (in 24.44 % of pp.)
For advanced evaluation...
Opt. end rating reached: 9  (in 20.00 % of pp.)
Opt. max rating reached: 11  (in 24.44 % of pp.)

Length of sol.    40.60       30.40         7         129
Length of OSs      8.00        0.00         8           8

Number of operators for each planning process:             Proportion-%
                                                          (of overall length)
 1                 0.00        0.00         0           0        0.00 %
 2                 0.02        0.15         0           1        0.05 %
 3                 0.00        0.00         0           0        0.00 %
 4                 3.76        3.44         0          14        9.25 %
 5                 3.71        2.46         0          10        9.14 %
 6                 4.13        3.29         1          12       10.18 %
 7                 3.27        2.22         1          11        8.05 %
 8                 3.87        2.90         1          12        9.52 %
 9                 0.07        0.25         0           1        0.16 %
10                 3.87        2.99         0          12        9.52 %
11                 2.36        1.85         1          11        5.80 %
12                14.58       13.51         0          56       35.91 %
13                 0.98        0.15         0           1        2.41 %

L.of del.chains    2.14        1.31         1           8
(Average number of dc.:  6.82 ---------> from it proportion-% : 14.78 %)

Numb.of tot.del.   2.89        3.08         0          13        7.12 %

Senseless visits in the
final solution     0.62        0.53         0           2
Numb.of inv.ops.   0.38        1.00         0           6        0.93 %
```

APPLICATION OF HEURISTICS (for each course of planning):

Minimizing the movement times				
Performance	1.90	0.30	1.20	2.77
Random	2.13	0.17	1.79	2.50
Maximizing the advantage of the drive by car				
Performance	1.96	0.89	1.00	4.33
Random	2.12	0.46	1.00	3.00
Considering priority				
Performance	1.74	0.25	1.25	2.40
Random	1.74	0.17	1.34	2.16
Considering urgency				
Performance	1.71	0.27	1.33	2.54
Random	1.97	0.17	1.63	2.39
Optimizing the waiting time				
Performance	1.40	0.17	1.11	2.13
Random	1.59	0.12	1.38	1.88
"Error"	0.27	0.57	0	2

ANALYSIS OF THE TIME COURSE (for each course of planning):

Too late	21.42 %	7.94 %	0 %	33 %
On time	60.91 %	7.43 %	48 %	79 %
Too early	3.46 %	4.59 %	0 %	19 %
T.e. [with W.]	14.21 %	5.70 %	5 %	25 %
Min. too early	24.94	20.18	3	89
Min. too late	29.18	17.79	10	97

First applied operator (proportion in %):

	In overall planning	In the final solution
1	0.00 %	0.00 %
2	2.22 %	0.00 %
3	0.00 %	0.00 %
4	28.89 %	46.67 %
5	8.89 %	2.22 %
6	28.89 %	44.44 %
7	8.89 %	0.00 %
8	2.22 %	0.00 %
9	4.44 %	0.00 %
10	13.33 %	4.44 %
11	0.00 %	0.00 %
12	2.22 %	0.00 %
13	0.00 %	0.00 %
Empty final solution		2.22 %

Human Subjects - PAD Task 5

```
EVALUATION OF PLANNING PROCESSES WITHIN PLAN-A-DAY
==================================================

Number of considered planning processes: 45
Number of optimum solutions            : 45
Average number of OSs for each PAD-task: 1.00

                  Mean        SD          Min         Max
-------------------------------------------------------------
End rating (Std.) 22.91        3.69        11          26    (OS%: 88.12 %)
Max rating (Std.) 23.71        2.64        15          26    (OS%: 91.20 %)
End rat. (Adv.)   21.91        4.65        11          26    (OS%: 84.27 %)
Max rat. (Adv.)   23.64        2.69        15          26    (OS%: 90.94 %)
Opt. Rating       26.00        0.00        26          26

For standard evaluation...
Opt. end rating reached: 17  (in 37.78 % of pp.)
Opt. max rating reached: 18  (in 40.00 % of pp.)
For advanced evaluation...
Opt. end rating reached: 17  (in 37.78 % of pp.)
Opt. max rating reached: 18  (in 40.00 % of pp.)

Length of sol.    43.62       27.61         7         124
Length of OSs      9.00        0.00         9           9

Number of operators for each planning process:             Proportion-%
                                                        (of overall length)
 1                 0.00        0.00         0           0        0.00 %
 2                 3.31        2.64         0          13        7.59 %
 3                 3.64        2.10         0          10        8.35 %
 4                 0.00        0.00         0           0        0.00 %
 5                 0.00        0.00         0           0        0.00 %
 6                 3.82        2.12         1          10        8.76 %
 7                 2.49        1.98         0           9        5.71 %
 8                 3.56        2.43         0          10        8.15 %
 9                 3.13        2.10         1          12        7.18 %
10                 4.09        2.71         1          12        9.37 %
11                 3.09        2.31         0          11        7.08 %
12                15.51       12.48         0          55       35.56 %
13                 0.98        0.15         0           1        2.24 %

L.of del.chains    2.18        1.49         1          11
(Average number of dc.:  7.11 ---------> from it proportion-% : 16.05 %)

Numb.of tot.del.   2.91        2.61         0          11        6.67 %

Senseless visits in the
final solution     0.18        0.38         0           1
Numb.of inv.ops.   0.80        1.67         0           8        1.83 %
```

APPLICATION OF HEURISTICS (for each course of planning):

Minimizing the movement times				
Performance	2.16	0.41	1.33	2.91
Random	2.39	0.16	2.06	2.67
Maximizing the advantage of the drive by car				
Performance	1.81	0.59	1.00	4.00
Random	2.56	0.34	1.50	3.00
Considering priority				
Performance	1.97	0.30	1.27	2.57
Random	2.01	0.13	1.76	2.24
Considering urgency				
Performance	2.06	0.46	1.36	3.40
Random	2.41	0.15	2.11	2.67
Optimizing the waiting time				
Performance	1.54	0.25	1.00	2.15
Random	1.68	0.14	1.39	1.91
"Error"	0.04	0.21	0	1

ANALYSIS OF THE TIME COURSE (for each course of planning):

Too late	11.73 %	8.45 %	0 %	35 %
On time	70.32 %	11.81 %	47 %	100 %
Too early	2.97 %	4.08 %	0 %	16 %
T.e. [with W.]	14.99 %	7.72 %	0 %	31 %
Min. too early	11.34	8.20	3	29
Min. too late	26.49	24.82	3	140

First applied operator (proportion in %):

	In overall planning	In the final solution
1	0.00 %	0.00 %
2	0.00 %	0.00 %
3	0.00 %	0.00 %
4	0.00 %	0.00 %
5	0.00 %	0.00 %
6	40.00 %	6.67 %
7	11.11 %	55.56 %
8	0.00 %	15.56 %
9	37.78 %	20.00 %
10	8.89 %	2.22 %
11	2.22 %	0.00 %
12	0.00 %	0.00 %
13	0.00 %	0.00 %
Empty final solution		0.00 %

EVA - PAD Task 4

```
EVALUATION OF PLANNING PROCESSES WITHIN PLAN-A-DAY
==================================================

Number of considered planning processes: 4500
Number of optimum solutions            : 4500
Average number of OSs for each PAD-task: 1.00

                  Mean        SD          Min         Max
-------------------------------------------------------------
End rating (Std.) 21.21        7.42         0          28    (OS%: 75.75 %)
Max rating (Std.) 25.51        2.59        11          28    (OS%: 91.12 %)
End rat. (Adv.)   21.21        7.43         0          28    (OS%: 75.75 %)
Max rat. (Adv.)   25.51        2.59        11          28    (OS%: 91.12 %)
Opt. Rating       28.00        0.00        28          28

For standard evaluation...
Opt. end rating reached: 1026  (in 22.80 % of pp.)
Opt. max rating reached: 1056  (in 23.47 % of pp.)
For advanced evaluation...
Opt. end rating reached: 1026  (in 22.80 % of pp.)
Opt. max rating reached: 1056  (in 23.47 % of pp.)

Length of sol.    40.47       29.50         5         208
Length of OSs      8.00        0.00         8           8

Number of operators for each planning process:             Proportion-%
                                                        (of overall length)
 1                 0.00        0.00         0           0        0.00 %
 2                 0.00        0.00         0           0        0.00 %
 3                 0.00        0.00         0           0        0.00 %
 4                 2.78        1.90         0          15        6.87 %
 5                 3.59        2.70         0          21        8.88 %
 6                 3.24        2.26         0          16        8.00 %
 7                 3.23        2.20         0          18        7.99 %
 8                 4.46        3.13         0          21       11.03 %
 9                 0.00        0.00         0           0        0.00 %
10                 3.47        2.23         0          17        8.58 %
11                 3.27        2.55         0          17        8.08 %
12                15.42       13.79         0          92       38.11 %
13                 1.00        0.00         1           1        2.47 %

L.of del.chains    2.72        1.19         1           6
(Average number of dc.:  5.67 ---------> from it proportion-% : 12.35 %)

Numb.of tot.del.   3.04        3.14         0          21        7.51 %

Senseless visits in the
final solution     0.00        0.05         0           1
Numb.of inv.ops.   2.29        2.72         0          19        5.65 %
```

APPLICATION OF HEURISTICS (for each course of planning):

Minimizing the movement times				
Performance	2.04	0.26	1.27	3.14
Random	2.14	0.19	1.68	2.81
Maximizing the advantage of the drive by car				
Performance	1.62	0.53	1.00	3.00
Random	2.10	0.35	1.00	3.00
Considering priority				
Performance	1.74	0.19	1.25	2.75
Random	1.78	0.16	1.22	2.23
Considering urgency				
Performance	1.74	0.29	1.25	2.50
Random	1.98	0.20	1.48	2.59
Optimizing the waiting time				
Performance	1.51	0.24	1.00	2.43
Random	1.58	0.12	1.19	2.14
"Error"	1.47	1.82	0	13

ANALYSIS OF THE TIME COURSE (for each course of planning):

Too late	17.44 %	6.86 %	0 %	40 %
On time	62.49 %	8.45 %	40 %	100 %
Too early	3.52 %	5.04 %	0 %	33 %
T.e. [with W.]	16.55 %	6.30 %	0 %	50 %
Min. too early	32.83	21.14	1	121
Min. too late	45.67	31.72	14	319

First applied operator (proportion in %):

	In overall planning	In the final solution
1	0.00 %	0.00 %
2	0.00 %	0.00 %
3	0.00 %	0.00 %
4	80.80 %	31.82 %
5	1.93 %	9.42 %
6	10.29 %	37.40 %
7	0.00 %	4.69 %
8	0.00 %	0.78 %
9	0.00 %	0.00 %
10	6.98 %	11.24 %
11	0.00 %	0.00 %
12	0.00 %	0.00 %
13	0.00 %	4.64 %
Empty final solution		0.00 %

EVA - PAD Task 5

```
EVALUATION OF PLANNING PROCESSES WITHIN PLAN-A-DAY
==================================================

Number of considered planning processes: 4500
Number of optimum solutions            : 4500
Average number of OSs for each PAD-task: 1.00

                  Mean        SD          Min         Max
-------------------------------------------------------------
End rating (Std.) 22.14        3.11         0          26    (OS%: 85.17 %)
Max rating (Std.) 22.94        1.43         9          26    (OS%: 88.24 %)
End rat. (Adv.)   22.12        3.23         0          26    (OS%: 85.06 %)
Max rat. (Adv.)   22.94        1.43         9          26    (OS%: 88.24 %)
Opt. Rating       26.00        0.00        26          26

For standard evaluation...
Opt. end rating reached: 284  (in 6.31 % of pp.)
Opt. max rating reached: 318  (in 7.07 % of pp.)
For advanced evaluation...
Opt. end rating reached: 284  (in 6.31 % of pp.)
Opt. max rating reached: 318  (in 7.07 % of pp.)

Length of sol.    44.86       41.21         5         393
Length of OSs      9.00        0.00         9           9

Number of operators for each planning process:             Proportion-%
                                                        (of overall length)
 1                 0.00        0.00         0           0        0.00 %
 2                 4.56        4.54         0          41       10.17 %
 3                 3.90        3.89         0          34        8.69 %
 4                 0.00        0.00         0           0        0.00 %
 5                 0.00        0.00         0           0        0.00 %
 6                 3.19        2.83         0          25        7.12 %
 7                 3.18        2.87         0          26        7.09 %
 8                 1.59        1.33         0          14        3.54 %
 9                 3.54        3.10         0          28        7.90 %
10                 3.54        3.11         0          27        7.89 %
11                 3.13        2.59         0          24        6.98 %
12                17.22       19.06         0         184       38.38 %
13                 1.00        0.00         1           1        2.23 %

L.of del.chains    2.49        1.07         1           7
(Average number of dc.:  6.92 ---------> from it proportion-% : 13.37 %)

Numb.of tot.del.   2.00        2.71         0          27        4.45 %

Senseless visits in the
final solution     0.01        0.08         0           1
Numb.of inv.ops.   1.23        1.84         0          18        2.73 %
```

APPLICATION OF HEURISTICS (for each course of planning):

Minimizing the movement times				
Performance	2.11	0.27	1.22	3.67
Random	2.25	0.17	1.75	2.83
Maximizing the advantage of the drive by car				
Performance	2.00	0.51	1.00	5.00
Random	2.42	0.29	1.50	3.00
Considering priority				
Performance	1.86	0.23	1.00	3.13
Random	1.87	0.13	1.45	2.34
Considering urgency				
Performance	2.24	0.43	1.38	4.00
Random	2.27	0.17	1.76	2.83
Optimizing the waiting time				
Performance	1.41	0.20	1.00	3.00
Random	1.47	0.14	1.07	2.07
"Error"	0.99	1.51	0	13

ANALYSIS OF THE TIME COURSE (for each course of planning):

Too late	18.28 %	10.29 %	0 %	41 %
On time	67.81 %	9.11 %	33 %	100 %
Too early	1.43 %	3.44 %	0 %	33 %
T.e. [with W.]	12.48 %	6.39 %	0 %	67 %
Min. too early	15.73	11.63	2	103
Min. too late	45.09	28.03	1	253

First applied operator (proportion in %):

	In overall planning	In the final solution
1	0.00 %	0.00 %
2	0.29 %	0.00 %
3	0.00 %	0.00 %
4	0.00 %	0.00 %
5	0.00 %	0.00 %
6	4.24 %	9.80 %
7	0.22 %	12.93 %
8	48.89 %	38.67 %
9	18.11 %	20.49 %
10	4.47 %	4.31 %
11	23.71 %	13.24 %
12	0.07 %	0.00 %
13	0.00 %	0.56 %
Empty final solution		0.00 %

Appendix D
Instructions for the Use of NWRun

Instructions for the Use of NWRun

Basic Concepts

NWRun is an application for the training of multi-layer networks. The present introduction to the use of NWRun expects that you are familiar with the basic concepts of connectionist networks in general and with multi-layer networks of the backpropagation-class in particular. For the theoretical foundations of the implemented learning algorithms see the respective literature. For example, the book by Zell (1997) provides a very good overview.

NWRun is a text-based application for the Win32 console, (therefore it runs under Windows 95/98/ME and Windows NT/2000). Most of the input and output is carried out by configuration files on the one hand, and log files on the other hand (all files are text files which can be viewed easily by the user). In addition, some parameters can be handed over to NWRun at the command line.

NWRun is based on a C++ class library. This library provides a network class, by which hierarchical Elman networks can easily be integrated in other C++ programs. Hierarchical Elman networks are a recurrent version of normal multi-layer networks of the backpropagation-class; when their recurrent features are disabled they work like normal multi-layer feed-forward networks (see Zell, 1997). While the network class library allows the integration of such networks in C++ programs, NWRun is a tool to train these networks in advance. Furthermore, the resulting networks can be used in every way desired. The weights are saved in a special file with a clearly defined structure. Both NWRun and the complete source code of the network library are available for download at ”www.wolframschenck.de/NetworkApplication.htm”.

User-defined input files

First, there is the configuration file (`projname.bpc`). `projname` is the name for the project which can be chosen by the user. In the configuration file the user specifies the basic architecture of the network, the type of training algorithm and its parameters.

Second, there is the data file (`dataname.bpd`). In this file the user specifies the input-output pattern pairs used both for training and testing (training and test data).

Output files

First, there is the log file (`projname.bpl`). This file contains important information about the network architecture, training algorithm, and the course of training. This file is intended to be read directly by the user.

Second, there is the indicator file (`projname.bpv`). In this file, all important indicators that are calculated during the course of training are written into a kind of table. This file can be imported in applications like Lotus 1-2-3, Microsoft Excel, or SPSS (e.g., by using these applications one can produce charts or graphs for certain indicators).

Third, there is the test file (`projname.bpt`). At the end of training, NWRun writes the detailed results of the network test into this file, if the user so wishes.

Fourth, there is the maximum test file (`projname.bpm`). At the end of training, NWRun writes the detailed results of the maximum test into this file, if the user so wishes.

Network files

First, there is the weight file that contains the network weights (`projname.bpw`). After a certain number of training epochs (the complete presentation of all pattern pairs together with the corresponding weight adjustments is called "epoch") and at the end of training, the network weights are saved into a file `projname_??.bpw`. The question marks indicate consecutive numbers. In this way, previous weight configurations are always accessible. In addition, the latest weights are always saved in `projname.bpw`.

Second, there is the parameter file that contains the current parameter settings for the chosen training algorithm, for the weight decay control, and for miscellaneous other purposes (`projname.bpp`). After a certain number of training epochs and at the end of training, these parameters are saved into a file `projname_??.bpp`. The question marks indicate consecutive numbers. In this way, previous parameter settings are always accessible. In addition, the latest parameter settings are always saved in `projname.bpp`.

Starting NWRun

When you start NWRun with "`NWRun /?`", the following screen appears:

```
** Program parameters of NWRun   [(c) 2000 by Wolfram Schenck] **
NWRun PROJNAME LEARN_EPOCHS DISPLAY_SCREEN DISPLAY_TEST
      LOAD_WEIGHTS SAVE_WP SNUM_START ENUM_START
      LOAD_PARA WRITE_LOG WRITE_SCREEN

** Standard values (If a numerical parameter is set to a negative number, **
**                  the standard value is used)                           **
PROJNAME      : Xor
LEARN_EPOCHS  : 1000
DISPLAY_SCREEN: 10
DISPLAY_TEST  : 0
LOAD_WEIGHTS  : Load weights from the unnumbered file
SAVE_WP       : Save only once at the end of training
SNUM_START    : Set automatically
ENUM_START    : Set automatically
LOAD_PARAMS   : Load parameters from the unnumbered file
WRITE_LOG     : 1
WRITE_SCREEN  : 1

Maximum number of hidden layers       : 10
Maximum number of shortcut connections: 30

** By pressing ESCAPE the training always can be terminated **
```

Arguments which are omitted are set to their standard values (except for LOAD_PARAMS). In the following, these command line arguments are explained more deeply.

LEARN_EPOCHS: Number of training epochs until NWRun terminates automatically.

DISPLAY_SCREEN: Number of training epochs after which the current training indicators are displayed on the screen (0 = never).

DISPLAY_TEST: Number of training epochs after which the current test indicators are displayed on the screen and written to the log file (0 = never).

LOAD_WEIGHTS: Number of the weight file from which the weights are loaded (0 = no weights are loaded; -1 = the weights are loaded from the unnumbered weight file).

SAVE_WP: Number of epochs after which the current weight and parameter configuration is saved (0 = save never; -1 = save only when NWRun terminates). If saving is enabled, it takes place even in the case of manual interruption of training caused by the user pressing the escape key.

SNUM_START: Number with which the numbering of the saved weight and parameter files begins (-1 = continue automatically).

ENUM_START: Number with which the numbering of the training epochs begins (-1 = continue automatically).

LOAD_PARAMS: Number of the parameter file from which the training parameters are loaded (0 = no parameters are loaded; -1 = the parameters are loaded from the unnumbered parameter file; omitted = the parameter file which has the same number as is chosen for the weight file is loaded).

WRITE_LOG: Determines, if the log file is written (1 = yes).

WRITE_SCREEN: Determines, if output is displayed on the screen (1 = yes).

Special Concepts

In the following, some special concepts of NWRun are explained more in detail. Otherwise, it would be very difficult for the reader to understand some of the parameter settings in the configuration file.

Copy layers

With NWRun one can train pure feed-forward multi-layer networks as well as a kind of multi-layer network with recurrent connections (namely, hierarchical Elman networks; refer to Zell, 1997). In hierarchical Elman networks, recurrent connections are realized by context layers (which are called copy-layers in NWRun). Copy layers can be assigned to every hidden layer

and to the output layer. They are always of the same size as the layer to which they are assigned (their basis layer). They work as a supplementary input: Each unit of the copy layer sends its activation to every unit of the basis layer. The activation of each unit in the copy layer, on the other hand, is determined by the sum of the activation of the corresponding unit in the basis layer in the preceding processing cycle (multiplied by a layer-specific theta and by a layer-specific term [1-lambda]) and of the activation of the copy layer unit itself in the preceding processing cycle (multiplied by a layer-specific lambda). Such recurrent connections extend the capabilities of multi-layer networks, so that they are able to recognize chronological information in series of input patterns.

Furthermore, the specification of copy layers must include information about their initialization. NWRun allows several modes of initialization: (1) Only once when NWRun starts; (2) In the beginning of every new training or test epoch; (3) Triggered by a specific signal in the training and test data. These modes are controlled by the parameters cl_mode and cl_trigger in the configuration file:

cl_mode = 0, cl_trigger = 0 $\Rightarrow$ Initialization mode (2)
cl_mode = 0, cl_trigger = 1 $\Rightarrow$ Initialization mode (3)
cl_mode = 1, cl_trigger = 0 $\Rightarrow$ Initialization mode (1)
cl_mode = 1, cl_trigger = 1 $\Rightarrow$ not defined

In initialization mode (3), the column at position (coInp+coOutp+1) in the data file is used for determining, if the copy layers are initialized again. If the value at this position equals 1, then the initialization takes place.

Automatic weight decay adaptation

The weight decay parameter *d*, which is often used in training algorithms of the backpropagation-class, can be automatically adapted by NWRun. This automatic adaptation procedure monitors the course of training. Whenever the reduction of the training error after a certain number of epochs fell short of a certain threshold, the decay parameter *d* is reduced by multiplication with a decay adaptation factor in the range [0.0; 1.0]. In NWRun, the precise behavior of the automatic adaptation is controllable through five parameters (in the configuration file).

First, wD_Adaptation determines the number of epochs on which the comparison of the training error is based (if wD_Adaptation equals zero, automatic weight decay adaptation is disabled). Second, to avoid dependence on single error values, which can be subject to sudden and short-lived changes, the training error at the beginning and at the end of the comparison period is not determined by a single value, but by the median of the error values of wD_MD epochs. Third, the threshold which cannot be transgressed, is defined by wD_DiffPercent (within the range]-∞; 1.0]). If the current error median is larger than the product of (1 – wD_DiffPercent) and the error median wD_Adaptation epochs ago, then an adaptation of the decay parameter *d* is carried out. This adaptation takes place by multiplication of *d* with the fourth parameter, wD_Factor

(within the range [0.0; 1.0]). And last, but not least, there is the initial value of *d*, represented by wDecay.

Remark: For the training algorithm BackPercolation no weight decay can be applied.

Description of the Files Used by NWRun

Configuration File

The configuration file has the following structure: Each line begins with a parameter name, followed by white space, followed by the parameter value. Blank lines are not allowed. Lines with comments must begin with the character "#". Deviations from this specification will cause NWRun to terminate with an error message. In the following, the parameters of the configuration file are explained.

Basic network architecture

DataFileName:	Name of the data file (with training and test data)
rangeExpand:	If equal to 1, the input and output values in the data file are automatically converted from the range [0;1] to the range [-1;+1].
nTrain:	Number of patterns in the data file which are used for training.
nTest:	Number of patterns in the data file which are used for testing
numLay:	Number of hidden layers (maximum value: 10)
coInp:	Number of input units (units of the input layer)
coOutp:	Number of output units (units of the output layer)
hidSizeN (N = 1..10):	Number of units in the N-th hidden layer

Shortcut connections

numSC:	Number of shortcut connections (maximum value: 30)
scNstart (N = 1..30):	Starting layer of the N-th shortcut connection (values: 0 = input layer, 1..numLay = hidden layers)
scNend (N = 1..30):	Ending layer of the N-th shortcut connection (values: 1..numLay = hidden layers, numLay+1 = output layer)

Copy layers

copyLay:	If equal to 1, copy layers are enabled.
lambdaN (N = 1..numLay+1):	lambda for the N-th layer (values for N: 1..numLay = hidden layers, numLay+1 = output layer) (values for lambdaN: 0.0 <= lambdaN <= 1.0; 0.0 ⇒ only memory for the last epoch; 1.0 ⇒ no reception of new information)

thetaN (N = 1..numLay+1): theta for the N-th layer (values for N: 1..numLay = hidden layers, numLay+1 = output layer) (values for thetaN [Copy factor]: 0.0 <= thetaN <= 1.0; to activate a specific copy layer, set its theta to a value larger than 0.0 [recommended: 1.0])

cl_value: Initialization value for the copy layer units. Normal range: From -1.0 [0.0] to 1.0. If cl_value is larger than +2.0, the units in the copy layers are set to random values in the range [-0.9; 0.9] or [0.1; 0.9], respectively.

cl_mode: See section "Special Concepts – Copy layers"

cl_trigger: See section "Special Concepts – Copy layers"

Network initialization

rand_down: Lower bound for the random initialization values of the network weights before training

rand_up: Upper bound for the random initialization values of the network weights before training

fixedWeights: If equal to 1, the weights are set either to rand_down or to rand_up.

Basic training specifications

alg: Type of training algorithm (alg = 1: Simple Backpropagation; alg = 2: Backpropagation with Momentum; alg = 4: ResilientPropagation; alg = 5: QuickPropagation; alg = 6: BackPercolation)

range: Range of unit activation (range = 0: unit activation between 0.0 and 1.0; range = 1: unit activation between -1.0 and 1.0)

errAmpl: If equal to 1, the error signal of the output layer is amplified by atanh (arcus tangens hyperbolicus).

AddGradSigm: Value which is added to the derivation of the activation function in the calculation of the gradient of the global error function (Standard value: 0.0; increasing this value may result in faster convergence or in instability of training).

wDecay: Initial decay parameter *d* (see section "Special Concepts – Automatic weight decay adaptation")

errMin: Minimum value for the training error. When this value is transgressed, NWRun terminates.

For all training algorithms except for ResilientPropagation and BackPercolation

gamma: Learning rate (often designated as η)

gamFit: If equal to 1, each weight matrix receives its own learning rate, which is gamma devided by the number of units in the sending layer.

For Backpropagation with Momentum

alpha: Momentum factor in the formula for determining the changes for "normal" weights

alpha_sc: Momentum factor in the formula for determining the changes for weights in shortcut connections

alpha_cl: Momentum factor in the formula for determining the changes for weights in connections coming from copy layers

For ResilientPropagation

DD0: Initialization value for the update values (Δ_0)

gam_minus: Reduction factor for the update values (η^-) (0 < gam_minus < 1)

gam_plus: Enlargement factor for the update values (η^+) (1 < gam_plus)

DD_min: Minimum for the update values (Δ_{min})

DD_max: Maximum for the update values (Δ_{max})

For QuickPropagation

eta: Maximum growth factor (μ) (recommended range: 1.01 to 1.2)

For BackPercolation

lambda0: Initial value for the amplification of the error signal of the output layer (called λ_0 oder $\lambda(1)$)

weightIniValue: Basis value for the initialization of weights (θ)

normType: The values 0 and 1 correspond to the norm types A and B.

normSqrt: If equal to 1, instead of the norm, its square root is used in the training algorithm.

bcOnline: If equal to 1, online training is carried out, otherwise offline training is performed.

nTrainReduce: If equal to or less than 1, no reduction of the training set is carried out in determining the training error. If nTrainReduce is larger than 1, the training set is reduced in determining the training error by nTrainReduce as divisor. In each epoch, the current subset is determined randomly.

Remarks: BackPercolation reacts in a very sensitive way towards different values for lambda0 and weightIniValue. For each problem, some testing in advance is necessary. Furthermore, the flags normType and normSqrt may be varied to obtain better results. bcOnline should normally be set to 1, since BackPercolation is designed for online training. nTrainReduce can be used to reduce training time; on the other hand, since only a subset of the training set is used for determining the current training error then, a loss of precision has to be taken into consideration.

Automatic weight decay adaptation

Remark: For an explanation of the parameters, see section "Special Concepts – Automatic weight decay adaptation".

Available Parameters: wD_Adaptation, wD_MD, wD_DiffPercent, wD_Factor

Additional evaluations and information about the course of training

writeIndicators: If larger than 0, the indicator file is created or continued. The indicators are saved after every writeIndicators epochs.

writeSpecialInd: If the indicator file is written and writeSpecialInd is larger than 0, every so many epochs (writeSpecialInd * writeIndicators), some specialized indicators are calculated and saved, as well.

writeTest: If equal to or larger than 1, the test file will be written after regular termination of NWRun (for the test data). If writeTest is equal to or larger than 2, two test files will be written after regular termination of NWRun, one for the test data and one for the training data.

MaximumTest: If equal to or larger than 1, the maximum test will be carried out during training and the maximum test file will be written after regular termination of NWRun (for the test data). If MaximumTest is equal to or larger than 2, the maximum test will be carried out during training for both test and training data, and after regular termination of NWRun two maximum test files will be saved, one for the test data and one for the training data.

Remark: The maximum test performs a comparison between the desired output and the actually produced output with regard to the rank order of output units concerning their activation values.

Data File

The data file has the following structure: Each line consists of numbers separated by white space. Blank lines are not allowed. Lines with comments must begin with the character "#". Deviations from this specification will cause NWRun to terminate with an error message.

Within each line, the first coInp numbers will be used for determining the activation of the input units. The following coOutp numbers will be used as desired output pattern. The number following behind is interpreted as trigger for the initialization of the copy layers (see section "Special Concepts – copy layers").

The first nTrain data lines in the data file are used for the training set, the following nTest lines constitute the set of test patterns. The network weights are adjusted by the chosen learning algorithm with regard to the training set. The test set is used for testing the generalization capabilities of the network.

Log File

The most important information in the log file is the course of the training error and the test error (the test error is written every DISPLAY_TEST epochs). The training error is the global error calculated for the patterns in the training set, the test error is the global error calculated for the test set. In NWRun, the global error is calculated according to the following formula:

$$E = \frac{1}{2 \cdot N_j \cdot N_p} \sum_j (o_{p,j} - t_{p,j})^2 \quad (1)$$

E is the global error the network produces, N_j is the number of output units, N_P is the number of patterns, $o_{p,j}$ is the actual activation of output unit j for pattern p, $t_{p,j}$ is the desired activation of output unit j for pattern p.

In addition, the log file shows the current weight decay term for each epoch and other parameters of the learning algorithms which can be subject to change during the course of training. Furthermore, the results of the maximum test (if enabled) are written every DISPLAY_TEST epochs.

Indicator File

If writing the indicator file is enabled, every writeIndicators epochs a subset of the following indicators is saved (column titles are given in parentheses; the subset depends on the chosen learning algorithm):

- The current epoch (Epoch)
- The current training error (ErrTrain)
- The current test error (ErrTest)
- The current linear training error (LinErrTrain). The global linear error is calculated by $E_{lin} = \frac{1}{2 \cdot N_j \cdot N_p} \sum_j \left| o_{p,j} - t_{p,j} \right|$. For explanation of the variables see equation (1).
- The current linear test error (LinErrTest)
- The current standard deviation of the linear training error (SDLinErrTrain)
- The current standard deviation of the linear test error (SDLinErrTest)
- The current value of the weight decay parameter *d* (wDecay)
- Value of gamma (gamma; only for Backpropagation and QuickPropagation)
- Value of eta (eta; only for QuickPropagation)
- Value of gam_minus (gamMinus; only for ResilientPropagation)
- Value of gam_plus (gamPlus; only for ResilientPropagation)
- Current amplification of the error signal of the output layer (lambda; only for BackPercolation)

Furthermore, every so many epochs (writeSpecialInd * writeIndicators), the following additional indicators are calculated and written saved in the indicator file (if this function is enabled):

- The average of the absolute values of the network weights (mvWeights)

- The standard deviation of the absolute values of the network weights (SD_Weights)
- The average of the update values (mvDD; only for ResilientPropagation)
- The standard deviation of the update values (SD_DD; only for ResilientPropagation)
- The average of the ratios between the update value and the absolute weight value for each connection (DD_Weight_ratio; only for ResilientPropagation)

The first line of the indicator file contains the column titles, separated by semicolons. In the following, each line corresponds to one epoch; within each line, the indicator values are separated by semicolons.

Weight File

The weight file is interesting for the user, if he wants to inspect or evaluate the weight configuration. Therefore a brief overview of the structure of the weight file is given, in the following. The weights are saved in form of matrices; each matrix represents the connections between two layers. The columns represent the units of the sending layer (from left to right), the rows represent the units of the receiving layers (from top to bottom). In standard connections from one layer to the next layer, the last column represents the connections coming from an additional on-unit (a unit with the fixed activation of 1.0, by which the threshold for each receiving unit is realized).

Within the weight file, the matrices for the standard connections between each layer and its following layer are saved first, beginning with the connections coming from the input layer. Afterwards, the weight matrices for the shortcut connections are saved in the same order as they are defined in the configuration file. Finally, the weights matrices for the connections between each copy layer and its basis layer are saved. They are only saved, if their corresponding theta parameter (`thetaN`) is larger than 0.0, and if copy layers are enabled in general.

Parameter File

In the parameter file, only the first value is interesting for the user. This value represents the current weight decay parameter *d*. By changing this value in the file `projname.bpp`, one can adjust *d* manually during the course of training. The parameter `wDecay` in the configuration file is only used for the first initialization of *d* when training begins. Changes of this parameter during the course of training have no effect.

Examples for Starting NWRun

For example, a project with the name `example` is considered. First, create the file `example.bpc` with the appropriate parameter settings for your project. In addition, you need a data file, for example `FirstAttempt.bpd`, which has to be declared in `example.bpc`. Imagine, that you want to

train your network for 1000 epochs, and that in every 10th epoch you want the training error to be displayed on the screen. In every 100th epoch, you wish the test error to be displayed as well. Since this is the first time training, no network weights are available for loading. At the end of training (and only then), you want the weights and training parameters to be saved to be able to continue training later on. For this purpose, start NWRun with the following command line:

```
NWRun example 1000 10 100 0 -1
```

Later on, when you want to continue the training of your example network, you must notify NWRun of loading the weights and training parameters:

```
NWRun example 1000 10 100 -1 -1
```

If you want NWRun to save the network weights and training parameters every 500th epoch, you should write the following command line instead:

```
NWRun example 1000 10 100 -1 500
```

When you actually try to perform these little examples, you will notice, that NWRun automatically maintains the correct consecutive numbers of epochs and weight and parameter files, even if training was disrupted manually by pressing the escape key.

Including a Network in Your Own C++ Program

With the network library, you can easily include a network that was trained with NWRun into your own C++ program. So far, that is only possible for users of Microsoft Visual C++ (by the way, it is most likely that only small changes in the source code are necessary in order to make it useable with alternative C++ compilers).

The network library consists of eight files: Netzwerk.cpp, nwMatrix.cpp, NWTest.cpp, NWTraining.cpp, StdAfx.cpp, Netzwerk.h, nwMatrix.h, StdAfx.h. You need these eight files to build a library you call best Network.lib. Afterwards, you only have to include Network.lib and Network.h in your own projects to be able to use trained networks in your own applications.

In the following, we start again from the `example` project. We assume, that the training of the network is finished and a weight file `example.bpw` is available. Then you can include the network by the following code fragment into your application:

```
include “netzwerk.h”

...

Netzwerk example; // This line creates a network with the variable name “example”
bool error;

error = example.ProjektInit( "c:\ExamplePath\example", 0, -1, false);
```

In the last line, the configuration file `example.bpc` in the folder “`c:\ExamplePath`” is loaded by the member function `ProjektInit`. The network `example` is initialized according to the

parameter settings in this file. (The first parameter of `ProjektInit` is the name and location of the project, not the name of the configuration file!) The second parameter of `ProjektInit` determines, that the weight file `example.bpw` is also loaded. A value of -1 would determine, that loading the weight file is omitted. A value larger than 0 would invoke loading a weight file with that specific number (`example_??.bpw`). The third parameter of `ProjektInit` is designated to the parameter file. The values for this parameter have the same meaning as for the preceding parameter for the weight file. Therefore, loading the parameter file is omitted in our example. The fourth parameter of `ProjektInit` is set to "false". This instructs `ProjektInit` not to load the data file.

After you have loaded your network in your application, you probably want to use it. This can be done through a code fragment such as:

```
Vector inp(8), outp(3); // Definition of the input and output vector
...
outp = example.UseNetwork( inp, 0 );
```

It is assumed in this example that the input layer of the network has eight units and the output layer three units. The input and output vector are defined according to this specification. The elements of an instance of the class `Vector` can be accessed through the following syntax: `vec(n)`. `vec(n)` is the n-th element of the vector `vec` (the elements are counted beginning with 1). The elements are of type `M_EL`. `M_EL` is by default equal to `float`.

In the last line of the preceding code fragment the vector of input units `inp` is transformed by the network `example` into the vector of output units `outp`. If the second parameter of `UseNetwork` is set to 1, the copy layers are reinitialized before the input pattern is propagated through the network. If the input vector does not have the correct size for the used network, the static member variable `NWError` (of the class `Netzwerk`) is set to "true". `NWError` can be checked by the member function `getNWError` (of the class `Netzwerk`). Thus, by the function `UseNetwork` you can utilize the networks you have trained by `NWRun` in your own C++ applications.

If you have further questions regarding the use of NWRun and the network library, please contact "mail@wolframschenck.de".