In J. A. Meyer, H. L. Roitblat, and S. W. Wilson, editors, Proc. of the 2nd International Conference on Simulation of Adaptive Behavior, pages 196-202. MIT Press, 1992.

# PLANNING SIMPLE TRAJECTORIES USING NEURAL SUBGOAL GENERATORS

Jürgen Schmidhuber
Department of Computer Science
University of Colorado
Campus Box 430
Boulder, CO 80309, USA
email: yirgan@cs.colorado.edu

Reiner Wahnsiedler
Institut für Informatik
Technische Universität München
Arcisstr. 21, 8000 München 2
Germany

**Abstract**

We consider the problem of reaching a given goal state from a given start state by letting an 'animat' produce a sequence of actions in an environment with multiple obstacles. Simple trajectory planning tasks are solved with the help of 'neural' gradient-based algorithms for learning *without a teacher* to generate sequences of appropriate subgoals in response to novel start/goal combinations.

*Relevant topic areas:* Problem solving and planning, goal-directed behavior, action selection and behavioral sequences, hierarchical and parallel organizations, neural correlates of behavior, perception and motor control.

## 1 INTRODUCTION

Many researchers in neuro-control and reinforcement learning believe that some 'compositional' method for learning to reach new goals by combining familiar action sequences into more complex new action sequences is necessary to overcome scaling problems associated with non-compositional algorithms.

The few previous ideas for attacking 'compositional neural sequence learning' are inspired by dynamic programming and involve reinforcement learning networks arranged in a hierarchical fashion (e.g. (Watkins, 1989), (Jameson, 1991), (Singh, 1992), see also (Ring, 1991) for alternative ideas).

Our approach is entirely different from previous approaches. It is based on some initial ideas presented in (Schmidhuber, 1991a). We describe *gradient-based* procedures for transforming knowledge about previously learned action sequences into appropriate subgoals for new problems. No external teacher is required. Our approach is limited, however, in the sense that it relies on differentiable (possibly adaptive) models of the costs associated with known action sequences.

## 2 A TYPICAL TASK

The following task is representative of a variety of analoguous tasks solvable by our method.

Consider figure 1. An 'animat' moves in the real plane defined by the $x$ and $y$ axis, producing a trajectory $\omega$ in $R^2$. There are obstacles in the form of circular swamps. As long as the 'animat' does not cross a swamp, there are no *costs*[1] ( = negative reinforcement). The $i$-th swamp $\Phi_i$ with

[1] It would be straight-forward, however, to introduce a term penalizing the length of $\omega$.

center $(x_i, y_i)$ builds the basis of a cone (growing in the third dimension) with tip $(x_i, y_i, -h_i)$. Crossing $\Phi_i$ costs

$$\int_{\omega} g(x, y, \Phi_i) \; dx \; dy, \tag{1}$$

where $g(x, y, \Phi_i) = 0$ if $(x, y)$ lies outside of $\Phi_i$, else $g(x, y, \Phi_i)$ is the distance between $(x, y)$ and $\Phi_i$'s cone (measured along the the line through $(x, y)$ perpendicular to the real plane). In figure 1, the grey area indicates the costs associated with a trajectory leading straight through a single swamp.

A problem $p$ is defined by a start state $s^p \in R^m$ and a goal state $g^p \in R^m$. In the above example, $m = 2$ – start states and goal states are simply given by pairs of cartesian coordinates. We are looking for an action sequence leading from $s^p$ to $g^p$ *with minimal costs.*

It is true that in theory such sequences could be learned by conventional reinforcement learning algorithms (e.g. (Barto, 1989), (Barto et al., 1983), (Anderson, 1986), (Schmidhuber, 1991b), (Sutton, 1984), (Lin, 1991), (Williams, 1988), (Watkins, 1989)). For the sake of argument, assume that the maximal step size of the 'animat' is just a tiny fraction of the obstacle diameter. Then all the above algorithms will take nearly forever to find appropriate cost-free trajectories for other than trivial start/goal combinations. One drawback of conventional algorithms is that they will try to learn each new task from scratch, instead of exploiting a possibility for speeding up learning and gaining efficiency by solving new tasks through *composition* of solutions for older tasks.

## 3 BASIC MODULES

Our approach is based on three modules.

The first module is a 'program executer' $C$, which may be a neural net (but does not have to be one). With a given problem $p$, $C$ emits a sequence of actions in response to its input vector, the 'problem name' $s^p \circ g^p$. Here '$\circ$' denotes the concatenation operator for vectors. We assume (1) that there are problems for which $C$ does not 'know' solutions with minimal costs but (2) that there also are many problems for which $C$ *does* 'know' appropriate action sequences (otherwise our method will not provide additional efficiency). $C$ may have learned this by a conventional learning algorithm – or possibly even by a recursive application of the principle outlined below.

The second module is the evaluator $E$. $E$'s input can be the concatenation $s \circ g$ of two states $s$ and $g$. $E$'s non-negative output $eval(s, g) \in R_0^+$ is interpreted as a prediction of the *costs* ( = negative reinforcement) for an action sequence (known by $C$) leading from $s$ to $g$. $eval(s, g) = 0$ means minimal expected costs.

$E$ represents a model of $C$'s current abilities. For the purposes of this paper, we need not specify the details of $E$ – it may be an adaptive network (like in (Schmidhuber, 1991a)) as well as any other mapping whose output is differentiable with respect to the input.

The third module is the module of interest: the *adaptive subgoal generator S*. $S$ is supposed to learn to emit a list of appropriate subgoals in response to a novel start/goal combination. Section 4 will present two architectures for $S$ – one for simultaneous generation of all subgoals, the other one for sequential generation of the subgoal list.

The $i$-th sub-goal of the list $(i = 1 \ldots n)$ is denoted by the vector $s^p(i) \in R^m$, its $j$-th component by $s_j^p(i)$. We set $s^p = s^p(0), g^p = s^p(n+1)$. Ideally, after training the subgoal-list $s^p(1), s^p(2), ..., s^p(n)$ should fulfill the following condition:

$$eval(s^p(0), s^p(1)) = eval(s^p(1), s^p(2)) = \ldots$$

$$\ldots = eval(s^p(n), s^p(n+1)) = 0. \tag{2}$$

Not all environments, however, allow to achieve (2). See section 5.

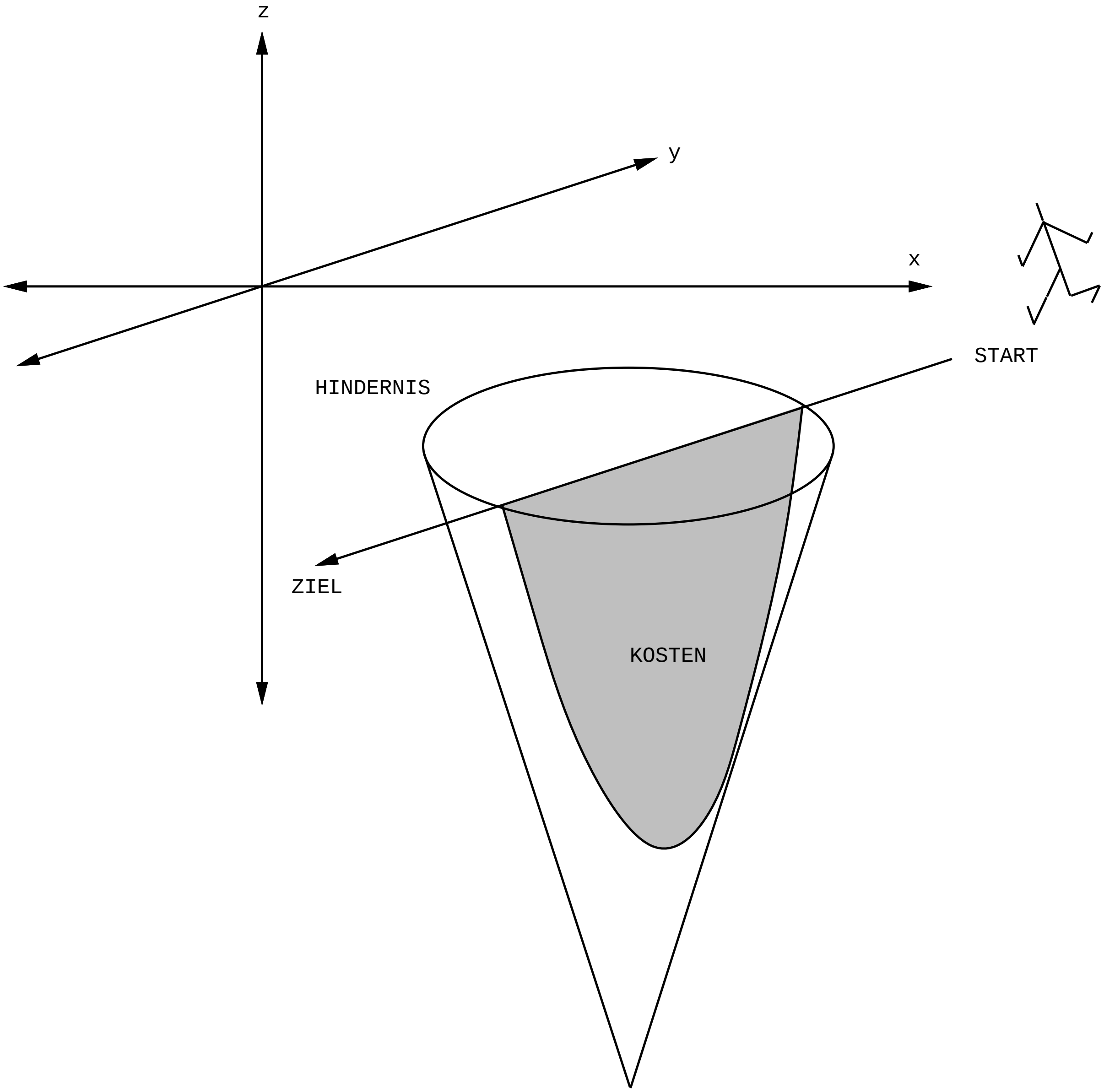


Figure 1: An ‘animat’ moving in the $x, y$ plane walks through a swamp. The costs are indicated by the grey area.

# 4 TWO SUBGOAL CREATING ARCHITECTURES

## 4.1 ARCHITECTURE 1

Figure 2 shows a static subgoal generator $S$ (a feed-forward back-prop net, e.g. (Werbos, 1974)). With problem $p$, the input vector of $S$ is $s^p \circ g^p$.

The output of $S$ is

$$s^p(1) \circ s^p(2) \circ \ldots \circ s^p(n).$$

$n+1$ copies of $E$ need to be connected to $S$ such that the input of the $k$-th copy of $E$ is equal to $s^p(k-1) \circ s^p(k)$. The output of the $k$-th copy of $E$ is $eval(s^p(k-1), s^p(k))$.

## 4.2 ARCHITECTURE 2

Figure 3 shows a *recurrent* subgoal generator $S$ (a back-prop net that feeds its output back to part of its input).

With problem $p$, the input vector of $S$ at the first 'time step' of the *sequential* subgoal generation process is $s^p \circ g^p$. The output of $S$ is $s^p(1)$.

At time step $t, 1 < t < n+1$, the input of $S$ is $s^p(t-1) \circ g^p$. Its output is $s^p(t)$.

Again we use $E$ to compute $eval(s^p(k-1), s^p(k)), k = 1, \ldots n+1$, from $s^p(k-1) \circ s^p(k)$.

# 5 OBJECTIVE FUNCTION

With both architectures we want to minimize

$$E^p = \sum_p \sum_{k=1}^{n+1} \frac{1}{2}(eval(s^p(k-1), s^p(k)))^2. \quad (3)$$

In words, we wish to find a sequence of subgoals such that the sum of the costs of all involved subprograms is minimized. This will be done by using gradient descent techniques to be described in the next section.

# 6 ALGORITHMS

With both architectures we apply the chain rule to compute the gradient

$$\frac{\partial \sum_{k=1}^{n+1} \frac{1}{2} eval^2(s^p(k-1), s^p(k))}{\partial W_S},$$

where $W_S$ denotes the weight vector of $S$. During each training iteration, $W_S$ has to be changed in proportion to this gradient.

With architecture 1, this is essentially done by back-propagating error signals (e.g. (Werbos, 1974), (Parker, 1985), (LeCun, 1985), (Rumelhart et al., 1986)) through copies of the evaluator modules down into the subgoal generator. Loosely speaking, each subgoal 'receives error signals from two adjacent copies of $E$'. These error signals are added and flow down into $S$, where they cause appropriate weight changes. One might say that in general two 'neighboring' evaluator copies (see figure 2) tend to pull their common subgoal into different directions. The iterative process stops when a local or global minimum of (3) is found. This corresponds to an 'equilibrium' of the partly conflicting forces originating from different evaluator copies.

The derivation of the more complex algorithm for the recurrent architecture 2 is analoguous to the derivation of conventional discrete time recurrent net algorithms (e.g. (Robinson and Fallside, 1987), (Williams, 1989), (Williams and Zipser, in press), (Schmidhuber, 1992)).

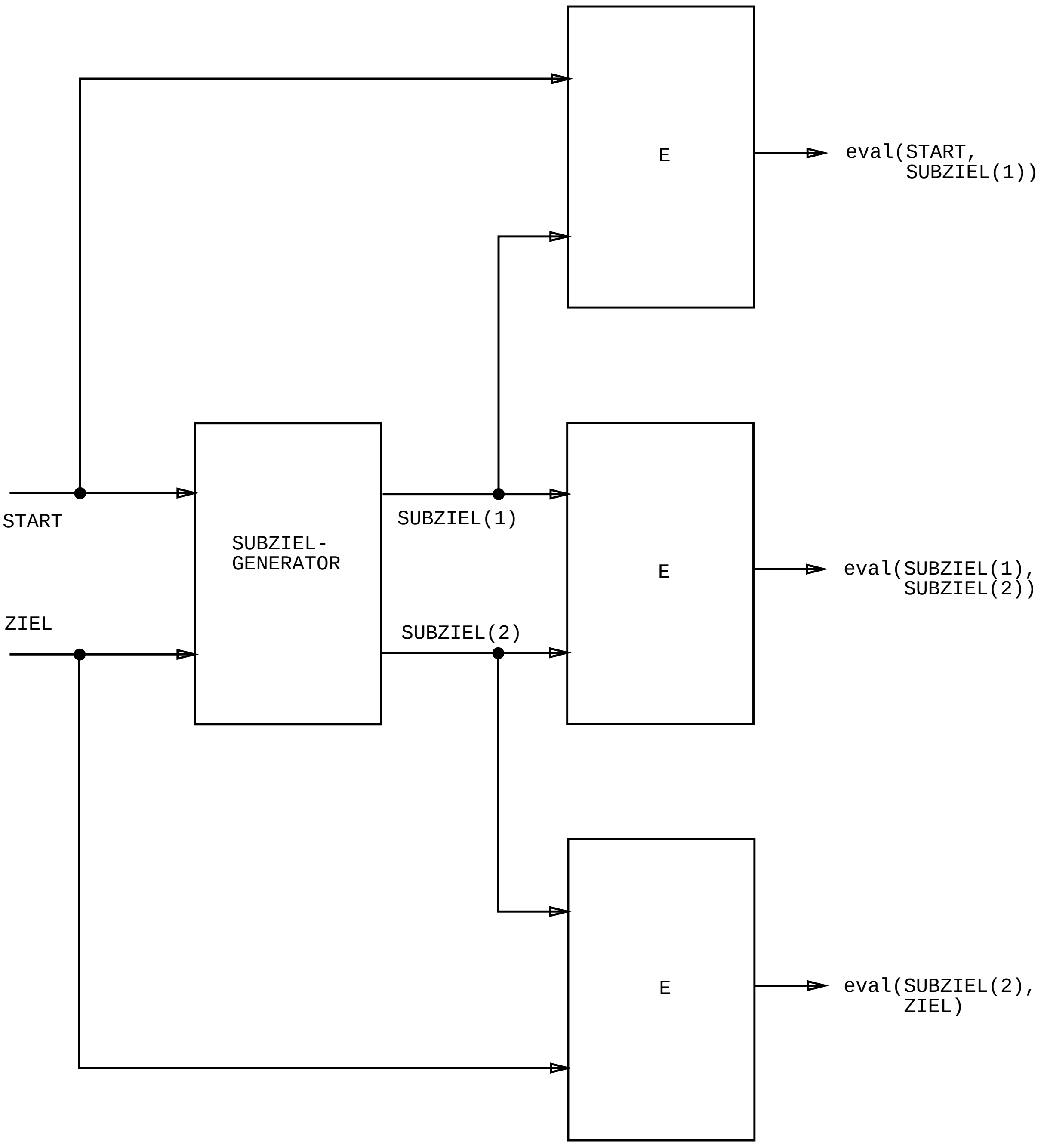


Figure 2: An adaptive non-recurrent subgoal generator emitting two subgoals. Three copies of the differentiable evaluation module are required to compute the proper gradient.

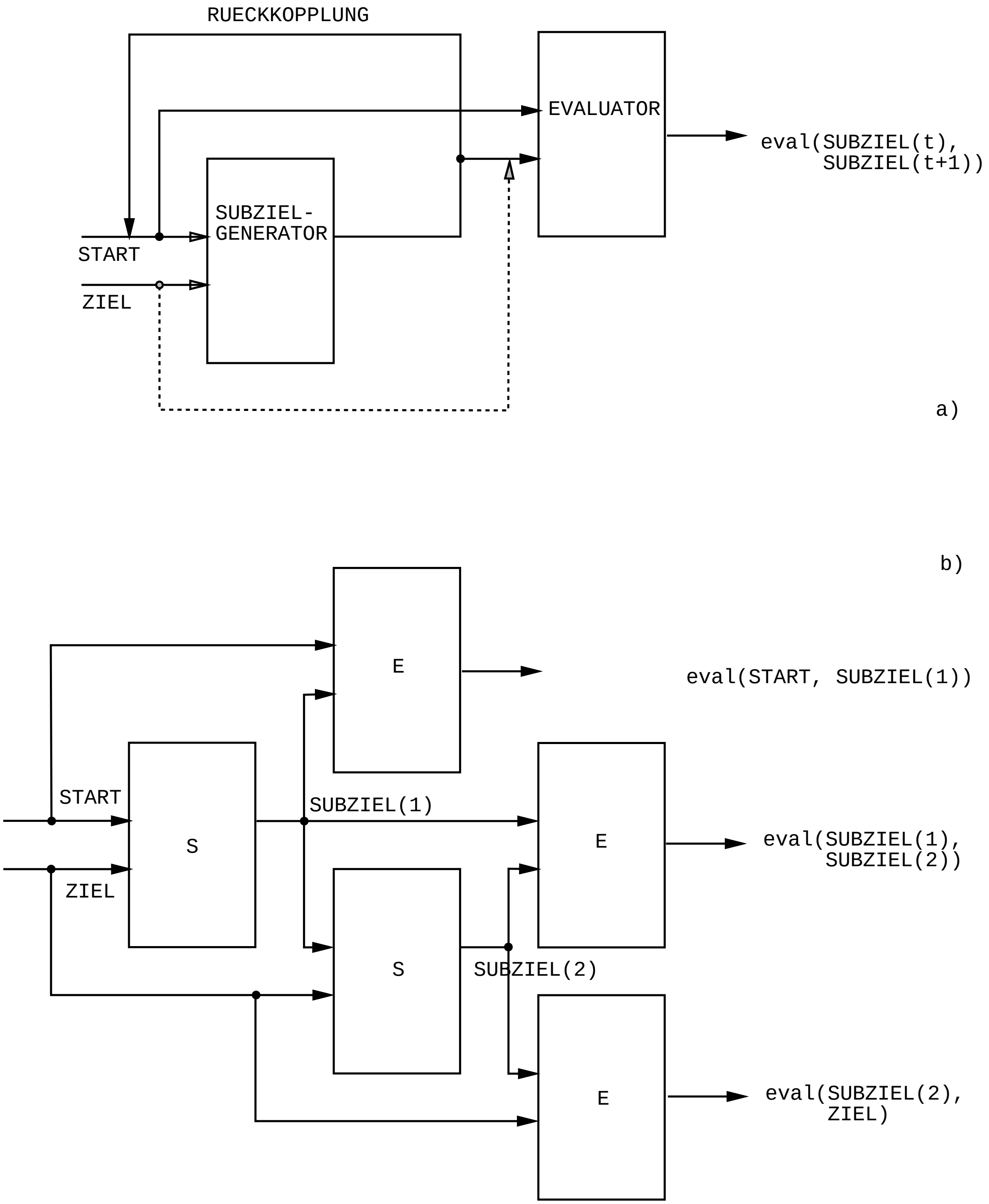


Figure 3: A recurrent subgoal generator emitting an arbitrary number of subgoals in response to a start/goal combination. Each subgoal is fed back to the START-input of the subgoal generator. The dashed line indicates that the evaluator needs to see the GOAL at the last step of the subgoal generation process. See text for details.

# 7 EXPERIMENTS

(Schmidhuber, 1991a) gives a simple example where the evaluator module $E$ itself is an adaptive back-prop network. In this section, however, we concentrate on the learning process of the subgoal generator $S$; the *eval* function and its partial derivatives are computed analytically.

For illustration purposes, we assume that $C$ 'knows' all possible action sequences leading to *straight* movements of the 'animat', and that the costs of all these action sequences are already known by $E$. In that case it is easy to compute (1). The start of the $k$-th 'sub-program' is $s^p(k) = (s^p_1(k), s^p_2(k))$, its end point is $s^p(k+1) = (s^p_1(k+1), s^p_2(k+1))$. (1) becomes equal to the area

$$F(s^p_1(k), s^p_2(k), s^p_1(k+1), s^p_2(k+1), \Phi_i) \tag{4}$$

defined by the trajectory of the 'animat' and the corresponding parabola-like projection onto the cone. See again figure 1.

For the $k$-th 'sub-program', *eval* is defined as

$$eval((s^p(k), s^p(k+1)) =$$

$$\sum_i F(s^p_1(k), s^p_2(k), s^p_1(k+1), s^p_2(k+1), \Phi_i). \tag{5}$$

Consider figure 4. A single swamp has to be overcome by the 'animat'. With 40 hidden nodes and a learning rate $\eta_S = 0.03$, a recurrent subgoal generator (architecture 2) needed 20 iterations to find a satisfactory solution.

Now consider figure 5. Multiple swamps separate the start from the goal. With 40 hidden nodes and a learning rate $\eta_S = 0.002$, a static subgoal generator (architecture 1) needed 22 iterations to find a satisfactory solution.

# 8 LIMITATIONS

*Generalization performance.* In most non-trivial cases, the approach did not generalize very well. After training $S$ on a range of different subgoal generation tasks (various randomly generated start/goal combinations), the subgoals emitted in response to previously unseen problems often were far from being optimal. More research needs to be directed towards improving generalization performance.

Another limitation of our approach has been mentioned above: It relies on differentiable (although possibly adaptive) models of the costs associated with known action sequences. The domain knowledge resides in these models – from there it is extracted by the subgoal generation process. There are domains, however, where a differentiable evaluator module might be inappropriate or difficult to obtain.

Even in cases where there is a differentiable model at hand the problem of local minima remains. Local minima did not play a major role with the simple experiments described above – with large scale applications, however, some way of dealing with suboptimal solutions needs to be introduced.

# ACKNOWLEDGEMENTS

Thanks to Mike Mozer for helpful comments on a draft of this paper. This research was supported in part by a DFG fellowship to J. Schmidhuber, as well as by NSF PYI award IRI–9058450, grant 90–21 from the James S. McDonnell Foundation, and DEC external research grant 1250 to Michael C. Mozer.

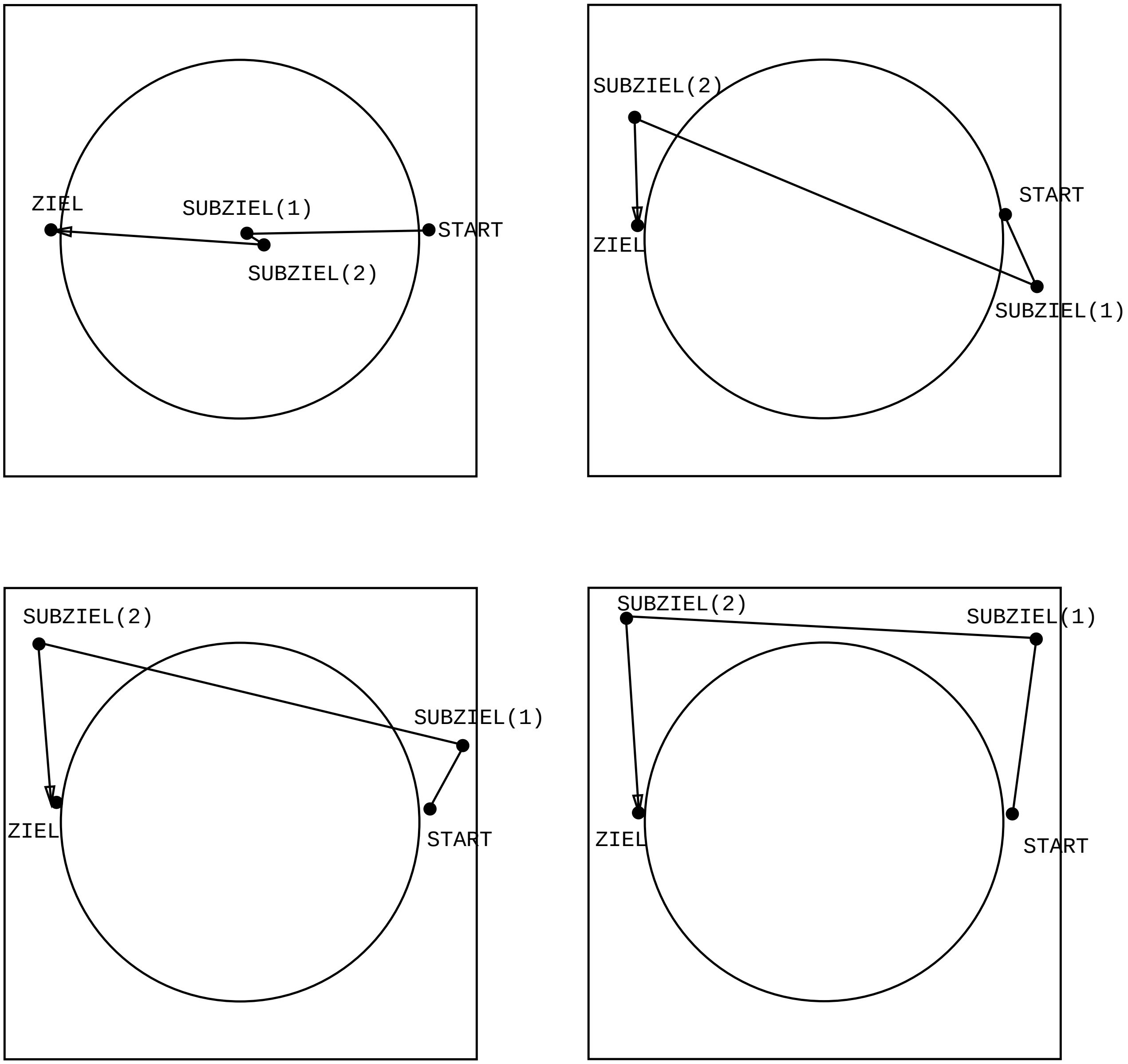


Figure 4: In this example, the task is to find a trajectory (composed of three 'sub-trajectories') leading from START to GOAL. The big circle represents a single swamp. The evolution of two subgoals emitted by an adaptive recurrent subgoal generator is shown.

# References

Anderson, C. W. (1986). *Learning and Problem Solving with Multilayer Connectionist Systems.* PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci.

Barto, A. G. (1989). Connectionist approaches for control. Technical Report COINS 89-89, University of Massachusetts, Amherst MA 01003.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846.

Jameson, J. (1991). Delayed reinforcement learning with multiple time scale hierarchical backpropagated adaptive critics. In *Neural Networks for Control.*

LeCun, Y. (1985). Une procédure d'apprentissage pour réseau à seuil asymétrique. *Proceedings of Cognitiva 85, Paris*, pages 599–604.

Lin, L. (1991). Self-improving reactive agents: Case studies of reinforcement learning frameworks. In Meyer, J. A. and Wilson, S. W., editors, *Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 297–305. MIT Press/Bradford Books.

Parker, D. B. (1985). Learning-logic. Technical Report TR-47, Center for Comp. Research in Economics and Management Sci., MIT.

Ring, M. B. (1991). Incremental development of complex behaviors through automatic construction of sensory-motor hierarchies. In Birnbaum, L. and Collins, G., editors, *Machine Learning: Proceedings of the Eighth International Workshop*, pages 343–347. Morgan Kaufmann.

Robinson, A. J. and Fallside, F. (1987). The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press.

Schmidhuber, J. (1991a). Learning to generate sub-goals for action sequences. In Kohonen, T., Mäkisara, K., Simula, O., and Kangas, J., editors, *Artificial Neural Networks*, pages 967–972. Elsevier Science Publishers B.V., North-Holland.

Schmidhuber, J. (1991b). Reinforcement learning in Markovian and non-Markovian environments. In Lippman, D. S., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3*, pages 500–506. San Mateo, CA: Morgan Kaufmann.

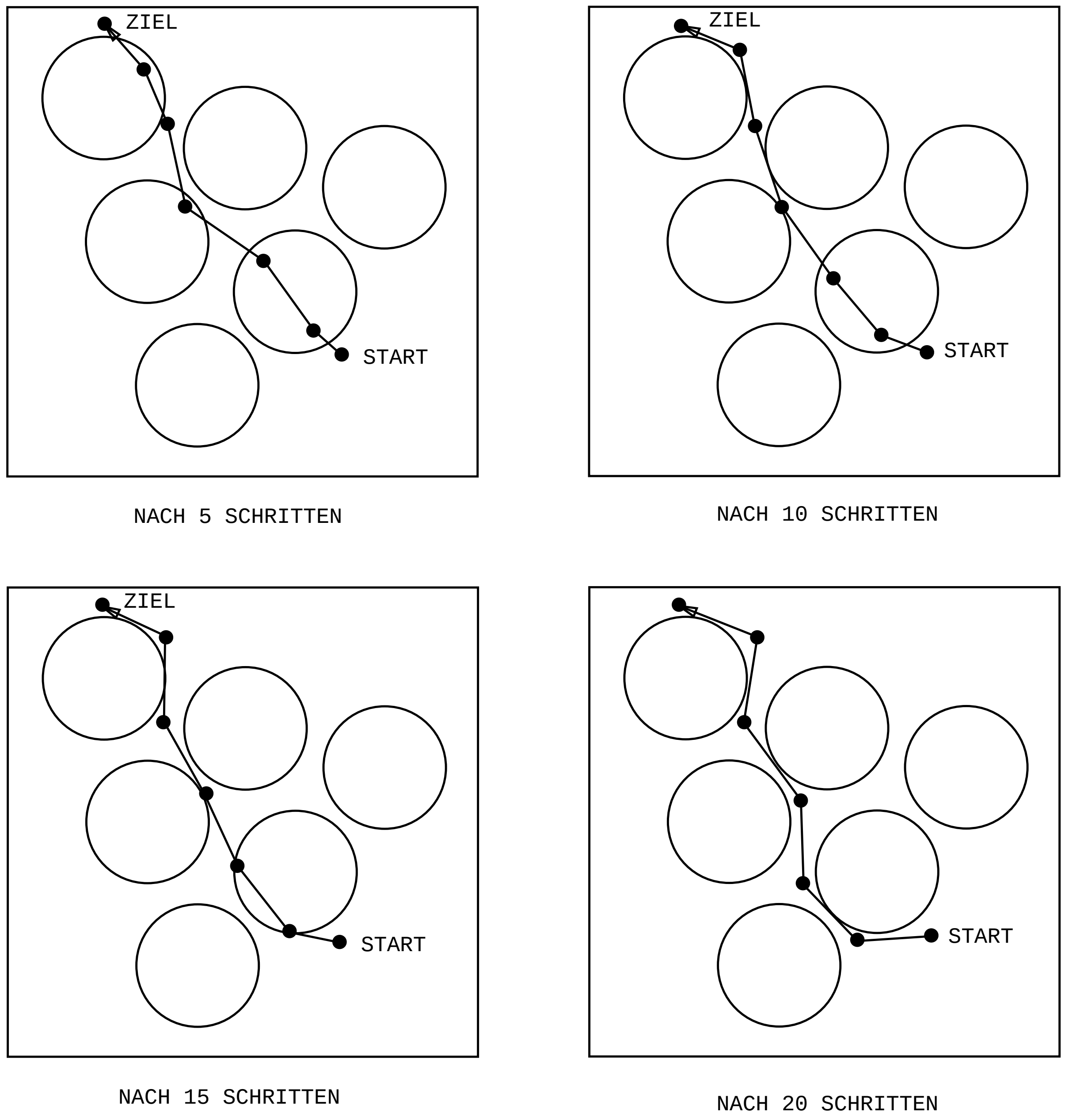
Schmidhuber, J. (1992). A fixed size storage $O(n^3)$ time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, 4(2):243–248.

Singh, S. (1992). The efficient learning of multiple task sequences. In Moody, J., Hanson, S., and Lippman, R., editors, *Advances in Neural Information Processing Systems 4*, pages 251–258, San Mateo, CA. Morgan Kaufmann.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci.

Watkins, C. (1989). *Learning from Delayed Rewards.* PhD thesis, King's College, Oxford.

Werbos, P. J. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University.



NACH 5 SCHRITTEN

NACH 10 SCHRITTEN

NACH 15 SCHRITTEN

NACH 20 SCHRITTEN

Figure 5: Here many swamps separate the START location from the GOAL location. The evolution of five subgoals (represented by little black dots) emitted by a non-recurrent subgoal generator is shown.

Williams, R. J. (1988). Toward a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, College of Comp. Sci., Northeastern University, Boston, MA.

Williams, R. J. (1989). Complexity of exact gradient computation algorithms for recurrent neural networks. Technical Report Technical Report NU-CCS-89-27, Boston: Northeastern University, College of Computer Science.

Williams, R. J. and Zipser, D. (1992). Gradient-based learning algorithms for recurrent networks and their computational complexity. In *Back-propagation: Theory, Architectures and Applications*. Hillsdale, NJ: Erlbaum.