Genetic Programming with Smooth Operators for Arithmetic Expressions: Diviplication and Subdition

Rasmus K. Ursem
EVALife Group, Dept. of Computer Science
Ny Munkegade, Bldg. 540, University of Aarhus
DK-8000 Aarhus C, Denmark
www.evalife.dk ursem@daimi.au.dk

Thiemo Krink
EVALife Group, Dept. of Computer Science
Ny Munkegade, Bldg. 540, University of Aarhus
DK-8000 Aarhus C, Denmark
www.evalife.dk krink@daimi.au.dk

Abstract - This paper introduces the smooth operators for arithmetic expressions as an approach to smoothening the search space in Genetic Programming (GP). Smooth operator GP interpolates between arithmetic operators such as $*$ and /, thereby allowing a gradual adaptation to the problem. The suggested approach is compared to traditional GP on a system identification problem.

I. Introduction

A major challenge in using evolutionary computation and other iterative search algorithms is the definition of the search space and, in particular, the neighborhood of a solution. This is straight-forward for numerical problems where the values range over a subset of $\mathbb{R}^n$, but can be a lot more complex for problems with no natural neighborhood, for instance, combinatorial problems such as the Traveling Salesman Problem (TSP). The standard approach to these problems is to define the neighborhood of a point p as the set of solutions reachable via an application of a simple operator. For example, the neighborhood of p in the TSP can defined as the paths obtainable by swapping two cities. The key issue in any neighborhood definition is to ensure that neighboring solutions are sufficiently related. Otherwise, most, if not all, search algorithms may experience difficulties in finding a good solution, because most of these algorithms rely on some degree of correlation between neighboring solutions.

In the early nineties Koza suggested Genetic Programming (GP) as an evolutionary approach to programming [4]. Genetic Programming is a special kind of Evolutionary Algorithm (EA) where the population consists of parse trees. This definition is indeed very general and GP has successfully been applied in a wide range of domains including time-series prediction, machine-coding, boolean logic, and many other areas. A simple, but particularly promising, application is to use GP to discover arithmetic expressions describing functional dependencies in measured data. In this case, the non-terminals in the parse trees are operators such as $\{+, -, *, /, sin, exp\}$ and the terminals are constants and variables related to the problem at hand. Defining a proper neighborhood in the search space of arithmetic functions is far from easy. Until now, most studies (if not all) define the neighborhood of an expression as the set of expressions reachable with a "minimal change" of the expression; for instance, exchanging a binary operator (e.g., $+$) with another binary operator (e.g., $*$). Although this is the smallest possible change[1] it can have quite a drastic effect on the expression. For example, changing $500 + x$ into $500 * x$ is a minimal change that may vastly change the genome's fitness. Hence, neighboring solutions in the search space may receive very different fitness values although their distance in the search space is minimal.

In the work presented here we introduce the so-called *smooth operators* for arithmetic expressions, which allow a more gradual change of a solution and thus a smoother fitness landscape. To our knowledge there is only one related approach suggested by Poli, Page, and Langdon [7], [8]. Their study is based on the idea of smoothening the search space of boolean parity problems by introducing a smoother representation of boolean operators. For this purpose they replace the standard operators `AND`, `OR`, and `NAND` by a subsymbolic representation where the output of a node is represented as a truth-table, i.e., `AND` is represented as `1000` and `OR` as `1110`. For instance, an

[1] Apart from modifying a constant.

in-between operator can be represented as `1100`. Combining this encoding with traditional bit-flip mutation and a specialized crossover operator allows the GP to interpolate between operators and thus search a smoother fitness landscape.

II. Smooth operators

The idea in the Smooth Operator GP (SOGP) is to combine multiple "ordinary" arithmetic operators into one and allow a gradual change from one operator to another through a number of parameters. In this paper we introduce the two smooth operators *diviplication* and *subdition*. Diviplication combines division and multiplication whereas subdition integrates subtraction and addition. Diviplication (DP) and subdition (SD) are defined as follows:

$$xDP[a,b]\,y \equiv sgn(x) * sgn(y) * |x|^a * |y|^b \quad (1)$$

$$xSD[a,b]\,y \equiv a * x + b * y \quad (2)$$

where $sgn(x)$ is the sign of x and $|x|$ is the absolute value of x. The slightly long definition of diviplication with $sgn(\cdot)$ and $|\cdot|$ is merely to avoid problems with undefined values of the power function such as $(-1)^{0.5} = \sqrt{-1}$. A more intuitive (but troublesome) definition of diviplication is:

$$xDP[a,b]\,y \equiv x^a * y^b \quad (3)$$

Defining diviplication according to equation (1) or (3) does not eliminate the ordinary operators division and multiplication – they are inherent in diviplication and can be obtained by setting a and b to 1 and -1.

$$\begin{array}{llll} a=1 & b=1 & : & xDP\,y \equiv x * y \\ a=1 & b=-1 & : & xDP\,y \equiv x/y \\ a=-1 & b=1 & : & xDP\,y \equiv y/x \end{array}$$

As mentioned, the goal in introducing the smooth operators is to allow a gradual change from one operator to another. For instance, a diviplication node with $a = 1$ and b changing gradually from 1 to -1 corresponds to a smooth change from $x * y$ to x/y. Some intermediate expressions are:

$$\begin{array}{llll} a=1 & b=1 & : & xDP\,y \equiv x * y \\ a=1 & b=0.5 & : & xDP\,y \equiv x * \sqrt{y} \\ a=1 & b=0 & : & xDP\,y \equiv x \\ a=1 & b=-0.5 & : & xDP\,y \equiv x/\sqrt{y} \\ a=1 & b=-1 & : & xDP\,y \equiv x/y \end{array}$$

Regarding subdition, the same considerations apply: The ordinary addition operator can be changed gradually into subtraction.

An important aspect of SOGP is setting the range and the step-size for a and b. The *order* of diviplication and subdition can be controlled by the range. For instance, setting it to [-2:2] allows diviplication expressions such as $x^2 * y$. The *discreteness* of the operator can be controlled by the step-size. For example, a step-size of 0.5 and a range of [-1:1] limits the possibilities to expressions having exponent a and b in $\{-1, -0.5, 0, 0.5, 1.0\}$. Hence, only expressions based on $x, \sqrt{x}, 1, 1/\sqrt{x}, 1/x$ and $y, \sqrt{y}, 1, 1/\sqrt{y}, 1/y$ are possible.

III. Experiments and Results

In our experiments we compared a traditional GP approach to the smooth operator GP approach. The pseudocode for both algorithms is listed in figure 1. In all experiments with algorithms, we used a population size of 100, elitism of 1 individual, max tree-depth of 3, probability of crossover $p_c = 0.5$, and probability of mutation $p_m = 0.5$. The number of generations was 2000 and the experiments were repeated 20 times. The mutation operator works as follows (u denotes an uniformly distributed number $u \sim U(0,1)$). First, for each individual the algorithm checked whether or not it should be mutated ($u < p_m$). If a mutation occured, then each mutation operator was applied if ($u < p_{mOP}$), where p_{mOP} is the probability of applying mutation operator OP. For both algorithms, we used the following mutation operators suggested by Angeline [2, Part C3.2.5]: Shrink ($p_{mshrink} = 0.1$), grow ($p_{mgrow} = 0.1$), switch ($p_{mswitch} = 0.1$), cycle ($p_{mcycle} = 0.1$), and Gaussian mutation of constants ($p_{mconst} = 0.6$). In addition to these operators, for the SOGP we used a Gaussian mutation operator on a and b in diviplication and subdition ($p_{mDPSD} = 0.6$). The variance in the Gaussian mutation operator was calculated by an $1/\sqrt{(1+g)}$ annealing scheme and multiplied by 0.1 times the length of the interval for the variable. The interval for constants was set to [-10:10] with a step-size of 0.01 and [-1:1] for parameter a and b in the smooth operators. In our experiments with the SOGP we used three step-sizes for a and b (0.5, 0.1, and 0.0).

A. The test problem

An interesting application of GP is system identification, where the objective is to discover the underlying model from a set of sample data. System identification is a key discipline in control engineering, because the design of the controller heavily relies on the presence of an accurate model. Figure 2 illustrates the interaction between the controller, the system, and the environment in such setups.

The change of a system state is usually modeled by a

Main
initialize population $P(0)$
evaluate $P(0)$
t=0
while(t<`maxgen`) {
t=t+1
Recombine $P(t-1) \rightarrow P'(t)$
Mutate $P'(t) \rightarrow P''(t)$
Evaluate $P''(t)$
Tournament selection $P''(t) \rightarrow P(t)$
}

Fig. 1. Pseudocode for the GP algorithm.

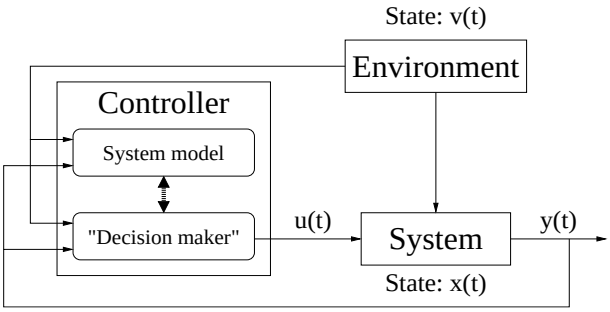

Fig. 2. The interaction between the controller, the system, and the environment.

number of difference equations of the form:

$$x_i(t+h) = x_i(t) + \Delta x_i(\mathbf{u}, \mathbf{x}, \mathbf{v}, t, h) \quad (4)$$

In the present study we used a complex and accurate model of a greenhouse to obtain a set of data for GP prediction. The simulator is described in details in [9][2], which is translated from the German version presented in [5]. An English summary of the German version can be found in[6]. The dataset contains values for the environment, the system state, and the control settings for four weeks of simulation in Germany in July. The data were sampled on a 15 minute basis ($h = 15$).

The greenhouse model is described by a number of non-linear differential equations, i.e., the function $\Delta x_i(\mathbf{u}, \mathbf{x}, \mathbf{v}, t, h)$ in equation (4) is a numerical approximation (Runge-Kutta) of the integral of the non-linear differential equations describing the system. The objective in this study was to find an arithmetic expression for Δx_{atemp} describing the indoor temperature at time $t+h$ as a direct function of the variables at time t, i.e., to approximate the integration of the differential equation by a simpler equation. The variables are listed in table I.

[2]Online at our website `www.evalife.dk`.

Our motivation for investigating this problem was twofold. First, we were interested in assessing the quality of the proposed smooth operators. In this context, the use of simulated data eliminates the problem of noise in measurements. Second, the dependencies in most real systems are described as non-linear differential equations, which cannot be solved explicitly. Hence, controller design often relies on simulation where the non-linear differential equations have to be approximated using numerical methods such as Runge-Kutta integration. This can be quite expensive in terms of CPU-time, thus, a sufficiently accurate approximation may speed up the evolutionary process by several magnitudes.

TABLE I
VARIABLES IN THE GREENHOUSE. THE VALUES IN THE "TYPE" COLUMN DENOTE THE TYPE (CONTROL, SYSTEM, OR ENVIRONMENT).

Variable	Description	Type	Used
u_{heat}	Heating	C	√
u_{vent}	Ventilation	C	√
u_{CO2}	CO2 injection	C	
u_{water}	Water injection	C	
x_{steam}	Indoor steam density	S	
x_{atemp}	Indoor air temp.	S	√
x_{CO2}	Indoor CO2 concentration	S	
x_{cond}	Indoor condensation	S	
v_{sun}	Outdoor sunlight intensity	E	√
v_{atemp}	Outdoor air temp.	E	√
v_{gtemp}	Outdoor ground temp.	E	
v_{RH}	Outdoor relative humidity	E	
v_{wind}	Outdoor wind speed	E	√
v_{CO2}	Outdoor CO2-level	E	

The set of terminals in the GP consist of the variables marked as used in table I, the hour of the day, and constants. The non-terminals in traditional GP were $\{+, -, *, /, sin, pow\}$, where / and pow are protected to prevent illegal numbers (e.g., division by zero). Likewise, the non-terminals for smooth operator GP were $\{DP, SD, sin, pow\}$. The operator $pow(x, y)$ was included in both sets to give traditional GP the same computational power as SOGP, but also to give SOGP the possibility of using the same complexity of power-expressions as traditional GP. The fitness was defined as the following sum of squared errors:

$$fit(X) = \frac{1}{N} \sum_{i=1}^{N} (\Delta x_{atemp}(i) - \Delta x'_{atemp}(i))^2$$

where Δx_{atemp} is the true change calculated from the data and $\Delta x'_{atemp}$ is the estimated change calculated by

TABLE II
MEAN AND STD. ERR. OF ALL 20 RUNS.

Algorithm	Training	Test
PRAG	0.1628	0.2089
GP	0.1026 ± 0.0291	0.1493 ± 0.0854
SOGP 0.5	0.1317 ± 0.0321	0.1838 ± 0.0518
SOGP 0.1	0.0940 ± 0.0289	0.1186 ± 0.0346
SOGP 0.0	0.0953 ± 0.0324	0.1271 ± 0.0464

the GP-expression X. The four weeks of data was split into a training set of three weeks and a test set of one week.

B. Results

The results from the experiments are presented in three different tables. Table II displays the mean and standard error calculated on all of the 20 runs. In Table III the best and the worst run are removed, i.e., the table displays the mean and standard error of 18 runs. Finally, Table IV displays the mean and standard error where the runs are grouped into the best 25%, the middle 50%, and the worst 25% of the runs. We present the data in this way because results obtained when just a few runs have really bad performance can be somewhat misleading. An algorithm may have excellent performance in 19 out of 20 runs, but just one run with poor performance will shift the mean upwards and hide the fact that the algorithm performed well in 95% of the runs.

A pragmatic approach to predicting the temperature change $\Delta x'_{atemp}$ is to assume that it only changes marginally between time t to time $t+1$. Hence, assuming $\Delta x'_{atemp} = 0$ may be a very competitive approach. The prediction error using this simple technique is listed in the tables under the label "PRAG". The traditional GP approach is labeled "GP" and Smooth Operator GP is denoted "SOGP". The step-size is written after SOGP, i.e., "SOGP 0.5" is Smooth Operator GP with step-size 0.5.

B.1 Discussion of the results

Tables II, III, and IV show that both the traditional GP and the three variants of SOGP are capable of obtaining an error less than the pragmatic approach on the training set, except for GP and SOGP 0.5 in the worst 25% of the runs (Table IV).

The comparison between GP and SOGP 0.5 on both the training and the test set is clearly in favor of GP, although the standard error intervals are overlapping. This could be because the SOGP has a different approach to minimizing the error. Since SOGP with a step-size of 0.5 is quite similar to traditional GP but cannot always switch a $*$ into a / in one mutation it may have a disadvantage.

TABLE III
MEAN AND STD. ERR. OF 18 RUNS EXCLUDING THE BEST AND THE WORST RUN.

Algorithm	Training	Test
PRAG	0.1628	0.2089
GP	0.1075 ± 0.0154	0.1421 ± 0.0276
SOGP 0.5	0.1389 ± 0.0094	0.1934 ± 0.0282
SOGP 0.1	0.0978 ± 0.0176	0.1239 ± 0.0194
SOGP 0.0	0.0987 ± 0.0222	0.1313 ± 0.0325

TABLE IV
MEAN AND STD. ERR. OF ALL 20 RUNS GROUPED INTO THE BEST 25%, THE MIDDLE 50%, AND THE WORST 25% OF THE RUNS.

	Algorithm	Training	Test
Best 25%	PRAG	0.1628	0.2089
	GP	0.0824 ± 0.0027	0.1055 ± 0.0025
	SOGP 0.5	0.1233 ± 0.0114	0.1559 ± 0.0142
	SOGP 0.1	0.0793 ± 0.0021	0.1005 ± 0.0043
	SOGP 0.0	0.0758 ± 0.0006	0.0995 ± 0.0047
Middle 50%	PRAG	0.1628	0.2089
	GP	0.1107 ± 0.0064	0.1430 ± 0.0134
	SOGP 0.5	0.1400 ± 0.0034	0.1918 ± 0.0171
	SOGP 0.1	0.0952 ± 0.0125	0.1214 ± 0.0118
	SOGP 0.0	0.0949 ± 0.0134	0.1232 ± 0.0172
Worst 25%	PRAG	0.1628	0.2089
	GP	0.1272 ± 0.0078	0.2358 ± 0.1364
	SOGP 0.5	0.1497 ± 0.0031	0.2325 ± 0.0061
	SOGP 0.1	0.1252 ± 0.0109	0.1547 ± 0.0078
	SOGP 0.0	0.1348 ± 0.0120	0.1879 ± 0.0211

The performance of GP vs. SOGP 0.1 and SOGP 0.0 is slightly in favor of SOGP on the training set, but a bit clearer on the test set when the best and worst runs are removed (table III). The 25-50-25 grouping (table IV) shows comparable performance in the best 25%, but favors SOGP in the middle 50% and the worst 25%.

Figure 3 illustrates the means of the prediction error (the fitness) for all algorithms on the training data. This graph supports the results displayed in the tables – GP, SOGP 0.1, and SOGP 0.0 are competitive with a slight advantage to the two SOGP algorithms. Finally, figure 4 shows the prediction error on the test data during the runs. Two interesting observations can be made from this graph. First, traditional GP seems to have a couple of

runs with poor performance during the entire first half of the optimization (up to generation 1200). Second, SOGP with step-size 0.1 does not have a single run with poor performance. This is also the case for SOGP 0.0 except for the short period around generation 1800.

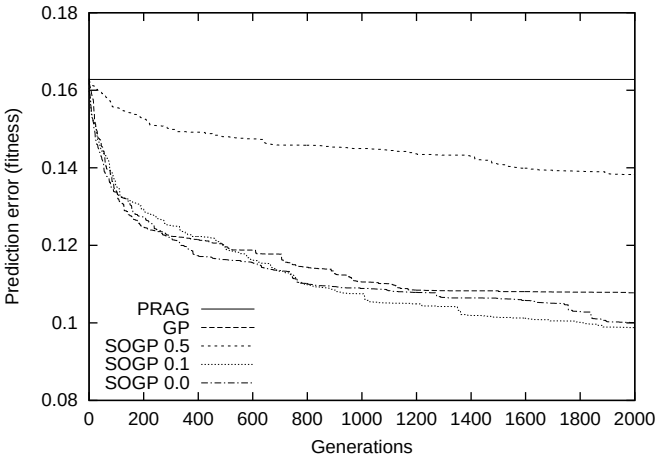

Fig. 3. The mean prediction error (fitness) on the training data during the optimization. Mean of 20 runs.

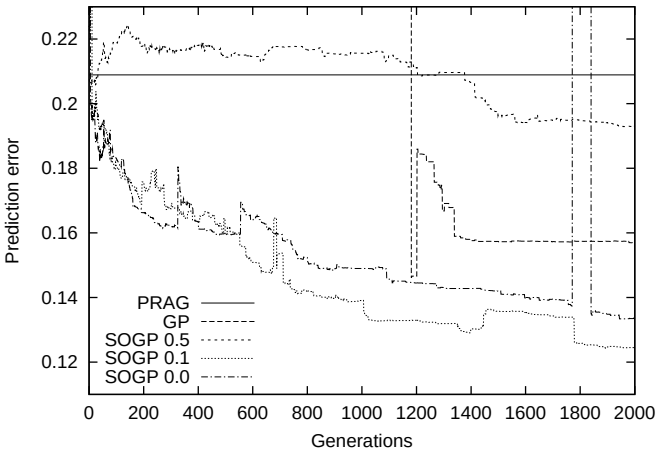

Fig. 4. The mean prediction error on the test data during the optimization. Mean of 20 runs.

In summary, traditional GP and SOGP with a small step-size both have a good performance with a slight advantage to the SOGP. The intermediate SOGP with step-size 0.5 is clearly not a good combination of these two extremes.

IV. Discussion

In this paper we introduced the smooth operators diviplication and subdition. The main objective of these novel operators is to allow a gradual change between ordinary arithmetic operators and thereby smoothen the space searched by the Genetic Programming algorithm. The approach was tested on a system identification problem involving prediction of the indoor temperature in greenhouses. The results from these experiments show a slight advantage to the smooth operator GP (SOGP) when comparing standard mean of all runs. Further analysis of the data revealed a clearer advantage to SOGP when results are grouped into three groups having the best 25%, the middle 50%, and the worst 25% of the runs.

Genetic Programming has been successfully applied to a large number of artificial benchmark problems in system identification (e.g., [1] and [3]). These studies showed that GP is a valuable approach to system identification. However, drawing conclusions from results of such artificial problems should be done with some scepticism in mind. The main problem is that the GP has an advantage in searching arithmetically defined target functions, i.e., the GP algorithm often searches the space of arithmetic expressions from which one solution produced the data used to calculate the fitness. Hence, it is possible to find that solution and achieve an error of 0.0. In real system identification problems this is rarely the case, because relationships can be more subtle than those expressible with simple arithmetics. Smooth operators may be well suited in this context, since the operators support a gradual adaptation to the data rather than the more discrete approach possible with ordinary GP.

The main drawback of smooth operators is the larger search space introduced by the additional variables a and b. However, this can be controlled by setting the step-size of the parameters. The effect of the search space size on the performance is currently an open question. The results presented in this paper suggest that a smaller step-size in fact has a positive effect on the performance. However, more experiments are clearly needed to investigate this further.

V. Future work

Introducing a novel technique usually raises more questions than can be answered in a single paper. The smooth operator GP for arithmetic expressions is no exception to this rule. First, additional experiments are needed to assess the usefulness on other system identification problems and problems in other domains. Second, in the present study we did not examine the importance of the order of the smooth operators (see section II). Some problems would clearly benefit from smooth operators with higher order (polynomials). Third, the experiments

on the variation of the step-size in the SOGP indicate that a smoother landscape is preferable; however, additional experiments are necessary to further investigate whether this is the case. Fourth, in this paper we only investigate diviplication and subdition. Other smooth operators could be introduced such as a combination of sinus and cosinus, which could be defined by $sin(\frac{\pi}{2}a+x)$. Furthermore, a combination of exp and ln may be worth investigating.

Acknowledgements

The authors would like to thank Bogdan Filipič (Dept. of Intelligent Systems, Jošef Stefan Institute, Ljubljana, Slovenia) for valuable comments and discussion.

References

[1] P. J. Angeline and D. B. Fogel. An evolutionary program for the identification of dynamical systems. In S. Rogers, editor, *Proceedings of SPIE: Application and Science of Artificial Neural Networks III*, volume 3077, pages 409–417, 1997. SPIE - The International Society for Optical Engineering.

[2] T. Bäck, D. B. Fogel, Z. Michalewicz, et al., editors. *Handbook of Evolutionary Computation*. IOP Publishing Ltd and Oxford University Press, 1997.

[3] T. Hatanaka and K. Uosaki. Hammerstein model identification method based on genetic programming. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 1430–1435, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 May 2001. IEEE Press.

[4] J. R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA, 1992.

[5] H. Pohlheim and A. Heissner. Optimale steuerung des klimas im gewächshaus mit evolutionären algorithmen: Grundlagen, verfahren und ergebnisse. Technical report, Technische Universität Ilmenau, 1996.

[6] H. Pohlheim and A. Heissner. Optimal control of greenhouse climate using real-world weather data and evolutionary algorithms. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1672–1677, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[7] R. Poli and J. Page. Solving high-order boolean parity problems with smooth uniform crossover, sub-machine code GP and demes. *Genetic Programming And Evolvable Machines*, 1(1/2):37–56, April 2000.

[8] R. Poli, J. Page, and W. B. Langdon. Smooth uniform crossover, sub-machine code GP and demes: A recipe for solving high-order boolean parity problems. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1162–1169, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[9] R. K. Ursem, T. Krink, and B. Filipič. A simulator for a tomato-producing greenhouse. Technical Report 2002-01, EVALife, Dept. of Computer Science, University of Aarhus, Denmark. Report online at www.evalife.dk, 2002.