You can judge a book by its cover – Evolution of gait controllers based on program code analysis

Jens Ziegler, Wolfgang Banzhaf
University of Dortmund, Dept. of Computer Science, D-44227 Dortmund, Germany

ABSTRACT

This paper introduces a Genetic Programming (GP) approach to automatically evolve control programs for walking robots. In contrast to earlier work, in which the evolution of gait control programs depended on the direct measurement of the quality of movements of simulated robots, in this paper a new method is presented that circumvents time consuming evaluations of control programs through the probabilistic evolutionary process.

1 INTRODUCTION

Evolution of control programs for walking robots with GP (1,2) – as every evolutionary experiment – requires the evaluation of individual solutions according to an explicitly defined fitness function. This in turn entails execution of the respective control program, observation of the generated movements of the robot for a certain period of time and calculation of the individuals' fitness values. Experiments have shown that genetic programming is able to automatically generate control programs that, when executed, make robots with arbitrary morphologies move by using their limbs or equivalent body parts[1] (3,4,5).

A typical evolutionary experiment, however, with its hundreds of generations and tens of individuals, quickly results in a protracted matter. Simulation, which would alleviate the problems caused by the use of real robots, implies other unwanted hitches and is very time consuming, once reasonably complex robots are to be simulated (6,7). In this contribution we show, how to circumvent the huge processing load when simulating behavioral programs.

The paper is organized as follows: the next section gives a short introduction to the use of surrogate functions in Evolutionary Algorithms (EA), Section 3 explains the linear representation of control programs for walking robots and its consequences for a meta-model approach. Section 4 introduces two different strategies for the use of meta-model assisted evolution, classi-

[1] The SIGEL simulation system for evolution of walking programs for arbitrary robot morphologies is an Open Source project and available via **http://sourceforge.net/projects/sigel**

fication and estimation. Section 5 shows results of experiments using the proposed strategy with two different robots, comparing the computational effort with reference experiments. Section 6 shows an example of a successful experiment using the estimation strategy.

2 SURROGATE FUNCTIONS

Reducing the number of evaluations during the evolutionary process without loss of quality at the same time is the goal of the method presented here. There already are a number of approaches to reduce the need for evaluations within the field of EC: if a fitness function is defined mathematically, an approximation may be computed by interpolation between sampling points (e.g. in (8,9)). In more complex cases, the fitness of an individual can be inherited from its ancestors (10). Within GP, there are approaches trying to reduce the number of evaluations by means of information theory and statistics (12). A comprehensive overview over the application of surrogate functions is presented e.g. in (11). The method proposed here follows a different approach:

Within a GP framework, tournament selection simply compares two (or more) individuals on the basis of their fitness values (e.g., in the case of walking robots used here, the walking speed). In order to decide which individuals represent superior walking programs, fitness values have to be determined before a decision can be made. In terms of Evolutionary Computation (EC), the decision is based on phenotypic information. In our approach, this evaluation of individuals, the primary factor for the overall runtime of the algorithm, is replaced by a decision, which is based only on the structure of the tournament's individuals, in EC terms called genotypic information. Once a decision is made, inferior individuals are replaced by the outcome of recombined and mutated individuals with superior quality. Thus, the classification of comparisons taking place during tournament selection acts as a surrogate function or a *meta-model* of the fitness function.

Earlier experiments have shown that a GP system is able to predict the outcome of a comparison of two individuals within a tournament, solely based on knowledge extracted from previously collected training data resulting in a 40% reduction of evaluations on average (13). GP was used in two forms here: (i) evolution to automatically produce better and better gait control programs, and (ii) evolution to generate better and better classifiers to discriminate between better and worse individuals of the first GP system in order to avoid their evaluation. Gait control programs were represented as a vector of integer values parameterizing an inverse kinematics, whereas evolved classifiers are linear lists of assembler-like instructions.

In this paper, the double use of GP remains the same, but the structure of individuals is more complex due to the fact that now linear GP is used to represent gait control programs, too.

3 META-MODEL STRATEGIES

The linear representation of computer programs within GP resembles assembler- or machine code programs. An example of a linear program is shown in Fig. 1. Each program of a GP population is a sequence of instructions. The following basic instructions are elements of the function set: ADD, SUB, MUL, DIV and MOD for arithmetic operations, COPY and LOAD

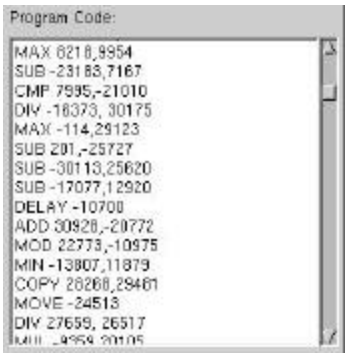

Figure 1: Part of a linear GP-individual used in the SIGEL system.

for register manipulation, CMP, JMP, DELAY and NOP for execution control, and the SENSE and MOVE command as instructions that are directly connected to the robot.
Execution of such linear codes causes the connected robot to move its joints according to the parameters of the MOVE statement. Feedback on the actual position and orientation of joints is gained with the SENSE command. In earlier experiments (13), a control program *p* is encoded as a vector of real values $x_i, i \in \{1,\ldots,n\}$,

$$p = \{x_1, \ldots, x_n\}\ , n=16, \tag{1}$$

parameterizing the inverse kinematic transformation that computes the trajectories of the foot points of each leg of the robot. Different parameters generate different trajectories, producing varying gait patterns, which are in turn subject of the selection and variation processes of the evolutionary algorithm. In the experiments described in this paper, a GP program *p* is encoded as a vector of real values $x_i, i \in \{1,\ldots,n\}$,

$$p = \{x_1,\ldots, x_n\}, n \in [2,\ldots,2 \cdot K \cdot 3], \tag{2}$$

if *p* is allowed to have at least one instruction and at most *K* instructions[2]. Switching from a low dimensional encoding of gait patterns to a linear representation thus has a large impact on the dimensionality of the search space (e.g. from a 16-dimensional, as in (13), to a 6000-dimensional search space, if *K*=1000). Any machine learning technique trying to predict the outcome of a comparison which is otherwise based on real fitness values and accomplished just by the operator ">" or "<" now has to search a high-dimensional search space in order to be able to compute the result of the comparison.

Training is done by evaluating programs from time to time. The quality of the prediction can then be measured by the fraction of correctly classified comparisons t_i based on the training set T:

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} g(t_i), \quad g(t_i) = \begin{cases} 1, \text{if classification of } t_i \text{ is correct} \\ 0, \text{else} \end{cases} \qquad \text{with } t_i \in T \tag{3}$$

[2] If a GP-program consists of *K* instructions with two parameters each, it can be represented as a vector in $(K \times 3)$-dimensional space.

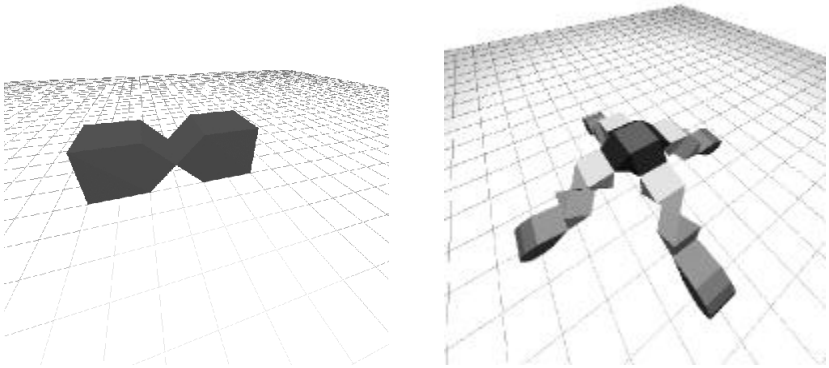

Figure 2: ***Left:*** **Simple robot consisting of two bodies connected by a rotational joint.** ***Right:*** **Four-legged robot with 12 degrees of freedom (DOF).**

Table 1: The Koza-tableau of parameter settings for the evolution of control programs

Parameter	Value
Fitness function	$f = \| x_{end} - x_{start} \| / t$ [m/s]
Crossover probability	0.67
Mutation probability	0.05
Reproduction probability	0.28
Minimum program length (instructions)	10
Maximum program length (instructions)	1024
Population size	100
Termination criterion	Time (12h/24h)
Simulation time	1 min.

Another, yet more complex method of circumventing time consuming evaluations is to not only predict the outcome of a comparison of two individuals (requiring just a "feeling" for superiority), but to predict the exact fitness values of programs. In this case, the quality of the prediction can be described by the sum of squared differences of the predicted and the actual fitness value. In the following sections, successful experiments with both methods are presented.

4 CLASSIFICATION OF CONTROL PROGRAMS

The proposed methods of either classification or fitness value prediction are applied to the evolution of control programs for two different types of robots (shown in Fig. 2). The results are compared with the average results of 10 reference experiments for both robots. The parameters of the reference experiment are shown in the Koza-Tableau in Tab. 1.

The very simple robot (Fig. 2, left) reaches an average a speed of 0.2 m/s, the maximum speed is in average 0.44 m/s. Within the time limit of 12 hours, the evolution reaches about 65 generations, during which 6500 programs were evaluated. In Fig. 3, the average and maximum fitness are shown. In the reference experiments with the four-legged robot (Fig. 2, right), the robots reach an average fitness of 0.08 m/s. Maximum value is on average 0.18 m/s. Within

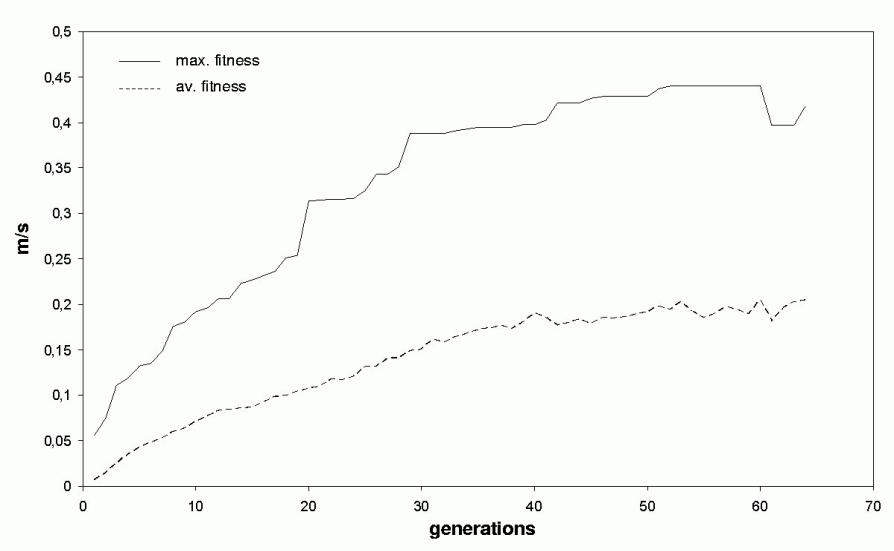

Figure 3: Average and maximum fitness of 10 runs of the unmodified SIGEL system with the simple robot. Parameters of the experiment are listed in Tab 1.

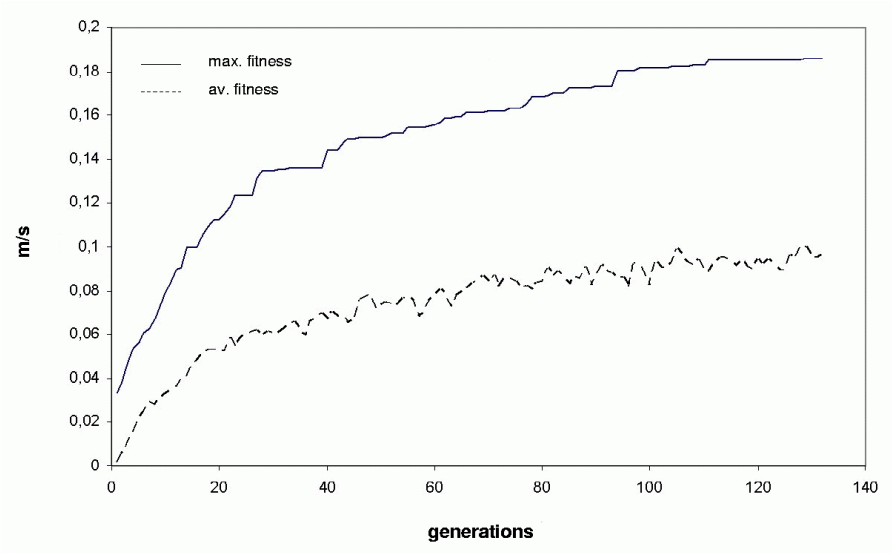

Figure 4: Average and maximum fitness of 10 runs of the unmodified SIGEL system with the four-legged robot.

24 hours, the evolution reaches approx. 130 generations. The development of the evolution is shown in Fig. 4.

In order to speed up evolution, time consuming evaluations should be replaced by faster classifications as often as possible. Furthermore, the quality of the evolved programs must be at least at the same level. Therefore, in an additional evolutionary process, programs are evolved, which are able to discriminate between programs with higher and programs with lower fitness values (the speed of the simulated robot, according to the fitness function given in Tab. 1.) Again, linear GP is used, with an augmented instruction set, which allows the evolved programs to scan the two gait control programs that act as input vectors. Real evaluations, taking

Table 2: The Koza-tableau of parameter settings for the meta-evolution of classifiers

Parameter	Value
Fitness function	Eq. (4)
Crossover probability	0.4
Mutation probability	0.36
Minimum program length (instructions)	0
Maximum program length (instructions)	250
Population size	8
Selection scheme	$(\mu,\lambda),\mu=8,\ \lambda=32$
Size of training set	50 cases
Tolerance threshold	40% (adaptive)

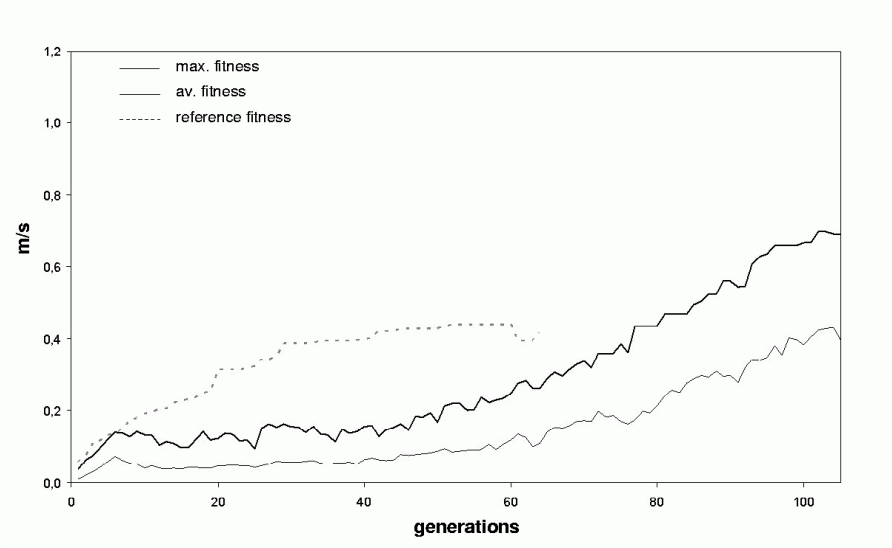

Figure 5: Average and maximum fitness of five runs with adaptive tolerance threshold.

place whenever the quality of the evolved classifying program falls below a certain threshold, build the training set, so that the quality of the classifiers can be computed according to Eq. (4). A threshold value of e.g. 40% means that only classifiers with less than 40% misclassification are used. Parameters of the meta-evolution of classifiers for both robots are listed in Tab. 2. Additionally, the tolerance threshold is adaptive, which means that the threshold decreases with time, creating a selective pressure during meta-evolution towards better and better classifiers.

Results of the experiment with the simple robot are shown in Fig. 5 and 6. The evolved programs in have an average fitness equal to the maximum fitness of the reference experiment. One reaches approx. 105 generations (compared to 65 in the reference experiment) with 12% less evaluations (5,800 compared to 6,600). Due to a high threshold at the beginning, development towards higher fitness values starts approx. at generation 30, when tolerance falls below 30%. Fig. 6 shows the average number of classifications per generation of five runs with adaptive threshold. It is clearly visible that with decreasing threshold the number of classifications decreases, too. In Fig. 7, the result of an evolutionary experiment with the four-legged robot and classification strategy is shown. In the reference run, approx. 135 generations were

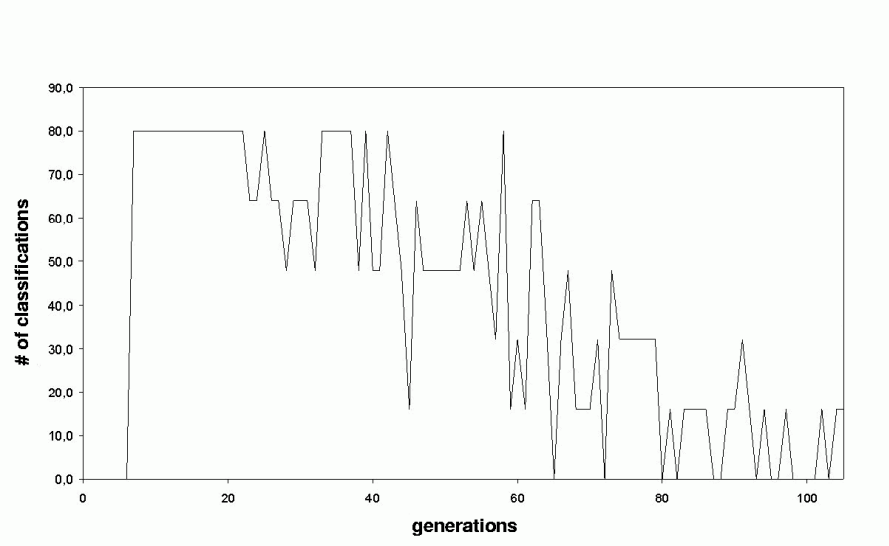

Figure 6: Average number of classifications per generation. A lower tolerance threshold induces fewer classifications. Again, a transition around generation 30 is visible.

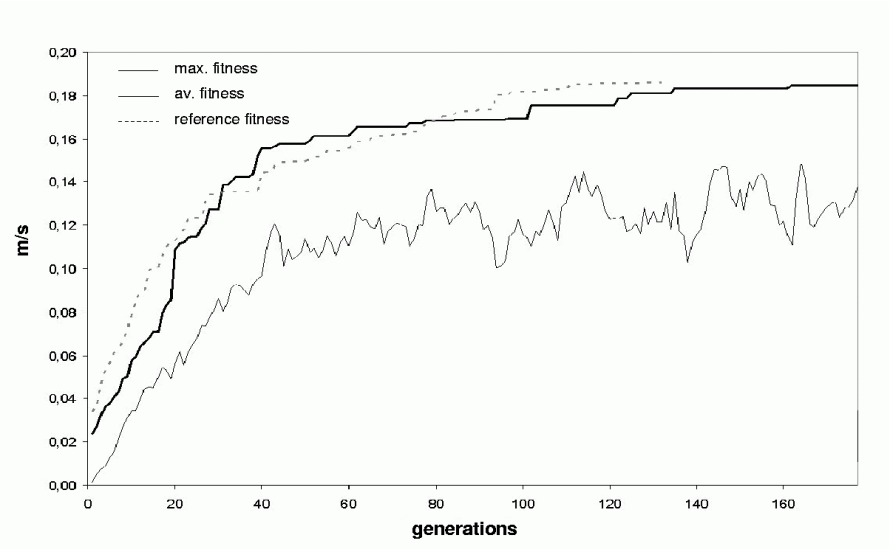

Figure 7: Average and maximum fitness of five runs, compared with the average maximum fitness of the reference experiments. Tolerance threshold is set to 25%.

evolved within 24 hours, using approx. 13,500 evaluations, reaching an average maximum fitness of 0.18 m/s. Using meta-evolution, 180 generations were evolved, but only 6,000 evaluations were used (saving more than 50%). The average maximum fitness of both experiments is nearly at the same level after 24 hours.

Classification of comparisons during tournament selection thus yields results with at least the same quality as experiments without any augmentation. It is remarkable that, in spite of the high-dimensional search space, classification of gait control programs is possible and successfully applicable. The advantages of this method increase, if the run time difference between classification and evaluation becomes larger.

5 FITNESS VALUE ESTIMATION

Evolution of GP programs that are able to predict the fitness values of gait control programs (i.e. to estimate the speed of the robot controlled by the respective program) is done with the same parameters as the evolution of classifiers (see Tab. 1), with the exception of the threshold value and the maximum program length. Threshold is set to a constant value of 0.05 here, which is the average fitness difference between gait control programs in the end of the reference experiment, whereas the fitness prediction programs are allowed to have 500 lines of code maximum. The average maximum fitness of four identical runs (0.2 m/s) is slightly better than the result of the reference experiment in Fig. 4 (0.18 m/s), with approx. 8% less evaluations (12,000 instead of 13,000). Evolution with fitness value estimation reaches 350 generations, whereas in the reference experiment only 130 generations were evolved.

It is an astounding result that evolved programs have shown the ability to predict the fitness values of gait control programs with an accuracy that is high enough to boost the evolutionary process.

6 CONCLUSION

In this paper, a GP system used to evolve gait control programs for two simulated robots – a simple 1 DOF robot and a more complex 12 DOF four-legged robot – is augmented by a second GP system, the latter evolving two different types of meta-models of the former system's fitness function. The classification strategy was successfully applied to evolve programs that are able to discriminate better and worse gait control programs, which are the result of the first GP system. The second variant – fitness value prediction – showed comparable performance to reference experiments of the un-augmented GP system. If time consuming evaluations can be replaced by faster classifications or fitness value estimations, a significant speed-up of the evolutionary process can be achieved.

The success of the proposed methods encourages the use of meta-model assisted evolution not only in genetic programming but also in other instances of evolutionary algorithms.

ACKNOWLEDGMENTS

This project is supported by the DFG (Deutsche Forschungsgemeinschaft), under grant
Ba 1042/6-2. The authors wish to thank Ralf Tenk and Marco Müller for their technical support.

REFERENCES

1. **J. R. Koza** (1992) *Genetic Programming*. MIT Press, Cambridge, MA
2. **W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone** (1998) *Genetic Programming -- An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco
3. **G. S. Hornby and M. Fujita and S. Takamura and T. Yamamoto and O. Hanagata** (1999) Autonomous Evolution of Gaits with the Sony Quadruped Robot. In W. Banzhaf

and J. Daida and A. E. Eiben and M. H. Garzon and V. Honavar and M. Jakiela and R. E. Smith., editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO*, pages 1297—1304, Morgan Kaufmann, San Francisco

4. **J. Busch, J. Ziegler, C. Aue, D. Sawitzki, A. Roß, W. Banzhaf** (2002) Automatic generation of control programs for walking robots using genetic programming. In J. Foster, E. Lutton, J.Miller, C. Ryan, A. Tettamanzi, editors, *Proceedings of the 5th European Conference on Genetic Programming (EuroGP)*, pages. 258—267, Springer, Berlin
5. **J. Ziegler, J. Barnholt, J. Busch, and W. Banzhaf** (2002) Automatic Evolution of Control Programs for a Small Humanoid Walking Robot. In P. Bidaud and F. Ben Amar, editors, *Proceedings of the 5th International Conference on Climbing and Walking Robots (CLAWAR)*, pages 109 –116, Professional Engineering Publishing, Bury St. Edmunds, UK
6. **N. Jakobi, P. Husbands, I. Harvey** (1995) Noise and the reality gap: the use of simulation in evolutionary robotics, In F. Moran, A. Moreno, J. Merelo, P. Chacon, editors, *Proceedings of the 3rd European Conference on Artificial Life (ECAL)*, pages 704—720, Springer, Berlin
7. **R. A. Brooks** (1992) Artificial Life and real robots, In In F. J. Varela and P. Bourgine, editors, Toward a Practice of Autonomous Systems: *Proceedings of the First European Conference on Artificial Life (ECAL)*, pages 3—10, MIT Press/Bradford Books, Cambridge, MA
8. **M. Emmerich, A. Giotis, M. Özdenir, T. Bäck, and K. Giannakoglou** (2002) Metamodel-assisted evolution strategies. In J.J. Merelo Guervos, P. Adamidis, H.-G. Beyer, J.-L. Fernandez-Villacanas, and H.-P. Schwefel, editors, *Proceedings of the Seventh International Conference on Parallel Problem Solving from Nature (PPSN)*. Springer, Berlin
9. **Y. Jin, M. Olhofer, and B. Sendhoff** (2002) A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation*, 6(5):481–494
10. **K. Sastry, D.E. Goldberg, and M. Pelikan** (2001) Don't evaluate, inherit. In L. Spector et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 551–558. Morgan Kaufmann, San Francisco
11. **Y. Jin** (2003) *Fitness Approximation in Evolutionary Computation - Bibliography*. http://www.softcomputing.de/amec.html. WWW-Document
12. **M. Giacobini, M. Tomassini, and L. Vanneschi** (2002) Limiting the number fitness cases in genetic programming using statistics. In J.J. Merelo Guervos, P. Adamidis, H.-G. Beyer, J.-L. Fernandez-Villacanas, and H.-P. Schwefel, editors, *Proceedings of the Seventh International Conference on Parallel Problem Solving from Nature (PPSN)*, Lecture Notes in Computer Science, LNCS 2439, pages 371–380. Springer, Berlin
13. **J. Ziegler, W. Banzhaf** (2003) Decreasing the Number of Evaluations in Evolutionary Algorithms by Using a Meta-Model of the Fitness Function. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, editors, *Proceedings of the 6th European Conference on Genetic Programming (EuroGP), LNCS 2610*, pages 264-175, Springer, Berlin
14. **I. Dahm, J. Ziegler** (2002) Using Artificial Neural Networks to Construct a Meta-model for the Evolution of Gait Patterns. In P. Bidaud and F. Ben Amar, editors, *Proceedings of the 5th International Conference on Climbing and Walking Robots (CLAWAR)*, pages 825 – 832, Professional Engineering Publishing, Bury St. Edmunds, UK

Learn about the saying You Can't Judge a Book By Its Cover, including its meaning and origin. Plus, see examples of this phrase.Â One should not form an opinion on someone or something based purely on what is seen on the surface, because after taking a deeper look, the person or thing may be very different than what was expected. Example: Thereâ€™s a new coworker at my job. At first, I thought he looked mean, like a tough guy that you wouldnâ€™t want to joke around with. But after talking to him, it turns out heâ€™s really nice and has a great sense of humor. This is a good example of the saying: â€œDonâ€™t judge a book by its cover.â€ Synonyms / Related Phrases: 1. Looks can be deceiving 2. Looks arenâ€™t everything 3. Things arenâ€™t The cover of a book is often the first interaction and it creates an impression on the reader. It starts a conversation with a potential reader and begins to draw a story revealing the contents within. But, what does the book cover say?Â This project is an improvised version of a CBIR (Content Based Image Retrieval) system. Once the user clicks a picture of a book-cover in real-time, we use a 3-Level matching system to retrieve closest matches of the book-cover and based on the retrieved book-cover it displays the summarised information about various aspects of the book that is obtained from sites such as Goodreads and Amazon.