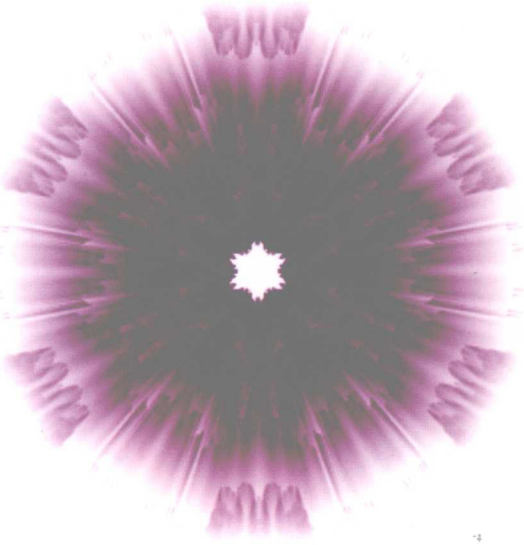

姜 群 著

Research on Design and Applications of Evolutionary Algorithms

进化算法的设计与应用研究

華中科技大學出版社
http://www.hustp.com

进化算法的设计与应用研究

Research on Design and Applications of Evolutionary Algorithms

姜 群 著

华中科技大学出版社
中国·武汉

本书涵盖了一系列重要的主题，范围从进化算法普遍涉及的问题（如算法参数控制与约束处理）到进化算法的研究热点——分布估计算法，并且重点突出分布估计算法的设计与应用。全书共有6章。第1章讨论如何设置进化算法各种参数的值，以这些参数值是否最好事先设置或在进化过程中如何改变等议题开始，给出了许多基于互补特征的不同方法的分类。第2章讨论使用进化算法时的约束处理。基于约束问题的分类，讨论从进化算法角度理解约束处理的含义，并研究了最常用的约束处理的进化技术。第3章集中于设计并行分布估计算法，更详细地给出了利用并行适应度评价和并行建模设计有效分布估计算法的具体指导。此外，第4章研究设计一类新的分布估计算法。最后，第5章和第6章汇合了分布估计算法解决医学和资源管理领域的优化问题的研究。

本书对进化算法领域的研究人员来说非常有用，也可供计算机专业的博士、硕士研究生使用。

图书在版编目(CIP)数据

进化算法的设计与应用研究 Research on Design and Applications of Evolutionary Algorithms/姜 群 著. —武汉：华中科技大学出版社，2010.7
ISBN 978-7-5609-6178-1

Ⅰ. 进… Ⅱ. 姜… Ⅲ. 遗传-算法 Ⅳ. TP18

中国版本图书馆 CIP 数据核字(2010)第 082178 号

进化算法的设计与应用研究
Research on Design and Applications of Evolutionary Algorithms 姜 群 著

策划编辑：谢燕群
责任编辑：熊 慧
封面设计：潘 群
责任校对：张 琳
责任监印：熊庆玉
出版发行：华中科技大学出版社(中国·武汉)
武昌喻家山 邮编：430074 电话：(027)87557437
录 排：武汉市兴明图文设计制作公司
印 刷：湖北新华印务有限公司
开 本：710mm×1000mm
印 张：6.75
字 数：150 千字
版 次：2010 年 7 月第 1 版第 1 次印刷
定 价：16.80 元

华中大版

本书若有印装质量问题，请向出版社营销中心调换
全国免费服务热线：400-6679-118 竭诚为您服务

Preface

This book covers broad spectrum important subjects ranging from general interests of EAs (such as algorithm parameter control and constraint handling) to the hottest topics in EAs—estimation of distribution algorithms (EDAs) with focusing on design and applications of EDAs.

The book is comprised of total of 6 chapters. In Chapter 1, we discuss how to set the values of various parameters of an evolutionary algorithm, beginning with the issue of whether these values are best set in advance or are best changed during evolution. Then, we provide a classification of different approaches based on a number of complementary features, and pay special attention to setting parameters on-the-fly. Then, we consider the issue of constraint handling using evolutionary algorithms in Chapter 2. Based on the classification of constrained problems, we discuss what constraint handling means from an EA perspective, and study the most commonly applied EA techniques to treat constraints. However, Chapter 3 focuses on the parallelization of EDAs. More specifically, it gives guidelines for designing efficient parallel EDAs that employ parallel fitness evaluation and parallel model building. Furthermore, techniques of implementations of new type of EDAs are studied in Chapter 4. Finally, Chapter 5 and Chapter 6 bring together some of EDAs approaches to optimization problems in the fields of medical science and resource management.

This book can be a useful and interesting tool for researchers working in the field of evolutionary algorithms. Also this book may be used by graduate students in computer science.

Qun Jiang
College of Computer Science and Engineering
Chongqing University of Technology

目　录

Contents

1 How to Set the Values of Parameters in EAs

1.1 Introduction

Evolutionary algorithms(EAs) are stochastic search methods that mimic the metaphor of natural biological evolution. The description of an evolutionary algorithm(EA) contains its components, thereby setting a framework while still leaving quite a few items undefined. For instance, a simple genetic algorithm(GA) might be given by stating it will use binary representation, uniform crossover, bit-flip mutation, tournament selection and generational replacement. For a full specification, however, further details have to be given, for instance, the population size, the probability of mutation p_m and crossover p_c, and the tournament size. These data—called the algorithm parameters or strategy parameters—complete the definition of the EA and are necessary to produce an executable version. The values of these parameters greatly determine where the algorithm will find an optimal or near-optimal solution, and whether it will find such a solution efficiently. Choosing the right parameter values is, however, a hard task.

Globally, we distinguish two major forms of setting parameter values: parameter tuning and parameter control. By parameter tuning we mean the commonly practiced approach that amounts to finding good values for the using these, which remain fixed during the run. Later on in this section we give arguments that any static set of parameters having the values fixed during an EA run seems to be inappropriate. Parameter control forms an alternative, as it amounts to starting a run with initial parameter values that are changed during the run.

Parameter tuning is a typical approach to algorithm design. Such tuning is done by experimenting with different values and selecting the ones that give the best results on the test problems at hand. However, the number of possible parameters and their different values means that this is a very time-consuming activity. Considering four parameters and five values for each of them, one has to test $5^4 = 625$ different setups. Performing 100 independent runs with each setup, this implies 62 500 runs just to establish a good algorithm design.

The technical drawbacks to parameter tuning based on experimentation can be

summarized as follows.

(1)Parameters are not independent, but trying all different combination systematically is practically impossible.

(2)The process of parameter tuning is time-consuming, even if parameters are optimized one by one, regardless of their interactions.

(3)For a given problem the selected parameter values are not necessarily optimal, even if the effort made for setting them was significant.

This picture becomes even more discouraging if one is after a "generally good" setup that would perform well on a range of problems or problem instances. During the history of EAs considerable effort has been spent on finding parameter values (for a given type of EA, such as GAs), that were good for a number of test problems. A well-known early example can be seen in *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*①, determining recommended values for the probabilities of single-point crossover and bit mutation on what is now called the De Jong test suite of five functions. About this and similar attempts②, it should be noted that genetic algorithms used to be seen as robust problem solvers that exhibit approximately the same performance over a wide range of problems③. The contemporary view on EAs, however, acknowledges that specific problems (problem types) require specific EA setups for satisfactory performance④. Thus, the scope of "optimal" parameter settings is necessarily narrow. There are also theoretical arguments that any quest for generally good EA, thus generally good parameter settings, is lost a priori, cf. the discussion of the No Free Lunch Theorem⑤.

---

① K A De Jong. An Analysis of the Behavior of a Class of Genetic Adaptive Systems. PhD Thesis, USA: University of Michigan, 1975, pp. 59-67.

② J J Grefenstette. Optimization of control parameters for genetic algorithms[J]. IEEE Transaction on Systems, Man and Cybernetics, 1986 (16):1, pp. 122-128.

J D Schaffer, R A Caruana, L J Eshelman, R Das. A study of control parameters affecting online performance of genetic algorithms for function optimization[C]//J D Schaffer. Proceedings of the 3rd International Conference on Genetic Algorithms. San Francisco: Moegan Kaufmann, 1998, pp. 51-60.

③ D E Goldberg. Genetic Algorithms in search, Optimization and Machine Learning[M]. UK: Addision-Wesley, 1989.

④ T Bäck, D B Fogel, Z Michalewicz, Eds. Evolutionary Computation [M]. Bristol: Institute of Physics Publishing, 1997.

⑤ D H Wolaert, W G Macready. No Free Lunch theorems for optimization[J]. IEEE Transactions on Evolutionary Computation. 1997(1):1, pp. 67-82.

To elucidate another drawback of the parameter tuning approach recall how we defined it: finding good values for the parameters before the run of the algorithm and then running the algorithm using these values which remain fixed during the run. However, a run of an EA is an intrinsically dynamic, adaptive process. The use of rigid parameters that do not change their values is thus in contrast to this spirit. Additionally, it is intuitively obvious, and has been empirically and theoretically demonstrated, that different values of parameters might be optimal at different stages of the evolutionary process①.

For instance, large mutation steps can be good in the early generations, helping the exploration of the search space, and small mutation steps might be needed in the late generations to help fine-tune the suboptimal chromosomes. This implies that the use of static parameters itself can lead to inferior algorithm performance.

A straightforward way to overcome the limitations of static parameters is by replacing a parameter p by a function $p(t)$, where t is the generation counter (or any other measure of elapsed time). However, as indicated earlier, the problem of finding optimal static parameters for a particular problem is already difficult. Designing optimal dynamic parameters (that is, functions for $p(t)$) may be even more difficult. Another possible drawback to this approach is that the parameter value $p(t)$ c hanges are caused by a "blind" deterministic rule triggered by the progress of time t , without taking any notion of the actual progress in solving the problem, i. e. without taking into account the current state of the search. A well-known instance of this problem occurs in simulated annealing where a so-called cooling schedule has to be set before the execution of the algorithm.

Mechanisms for modifying parameters during a run in an "informed" way were realized quite early in evolutionary computing(EC) history. For instance, evolution strategies changed mutation parameters on-the-fly by Rechenberg's 1/5 Success Rule using information on the ratio of successful mutations. Davis experimented within GAs with changing the crossover rate based on the progress and similar approaches is the presence of a human-designed feedback mechanism that utilizes actual information about the search process for determining new parameter values.

Yet another approach is based on the observation that finding good parameter

---

① T Bäck. The interaction of mutation rate, selection and self-adaptation within a genetic algorithm[C]//R Männer, B Manderick. Proceedings of the 2nd Conference on Parallel Problem Solving from Nature. North-Holland: Amsterdam, 1992, pp. 85-94.

values for an evolutionary algorithm is a poorly structured, ill-defined, complex problem. This is exactly the kind of problem on which EAs are often to use an EA for tuning an EA to a particular problem. This could be done using two EAs: one for problem solving and another one—the so-called meta EA—to tune the first one[①]. It could also be done by using only one EA that tunes itself to a given problem, while solving that problem. Self-adaptation, as introduced in evolution strategies for varying the mutation parameters, falls within this category. In the next section we discuss various options for changing parameters, illustrated by an example.

1.2 How to Change Parameters

Let us assume we deal with a numerical optimization problem to minimize

$$f(\overline{X})=f(x_1,x_2,\cdots,x_n)$$

subject to some inequality and equality constraints

$$g_i(\overline{X})\leqslant 0 \quad i=1,2,\cdots,q$$

and

$$h_j(\overline{X})=0 \quad j=q+1,q+2,\cdots,m$$

where the domains of the variables are given by lower and upper bounds

$$l_i\leqslant x_i\leqslant u_i \quad \text{for} \quad 1\leqslant i\leqslant n$$

For such a numerical optimization problem we may consider an evolutionary algorithm based on a floating-point representation, where each individual $\overline{x}$ in the population is represented as a vector of floating-point numbers $\overline{x}=\langle x_1,x_2,\cdots,x_n\rangle$.

1.2.1 Changing the Mutation Step Size

Let us assume that offspring for the next generation is produced by arithmetical crossover and Gaussian mutation, replacing components of the vector $\overline{x}$ by

$$x_i'=x_i+N(0,\sigma)$$

where $N(0,\sigma)$ denotes a random number drawn from a Gaussian distribution with zero mean and standard deviation σ. By using a Gaussian distribution here, small mutations are more likely than large ones. The particular feature of mutation and the very basis of self-adaptation in EA is that the step sizes are also included in the chromosomes and they themselves undergo variation and selection. The simplest

① B Friesleben, M Hartfelder. Optimization of genetic algorithms by genetic algorithms [C]//R F Albrecht, C R Reeves, N C Steele. Artificial Neural networks and Genetic Algorithms. New York: Springer, 1993, pp. 392-399.

method to specify the mutation mechanism is to use the same σ for all vectors in the population, for all variables of each vector, and for the whole evolutionary process, for instance, $x_i' = x_i + N(0, 1)$. As indicated by many studies①, it might be beneficial to vary the mutation step size. We shall discuss several possibilities in turn.

First, we can replace the static parameter σ by a dynamic parameter, i. e. a function $\sigma(t)$. This function can be defined by some heuristic rule assigning different values depending on the number of generations. For example, the mutation step size may be defined as

$$\sigma(t) = 1 - 0.9 \cdot \frac{t}{T}$$

where t is current generation number varying from 0 to T, which is the maximum generation number. Here, the mutation step size $\sigma(t)$, which used for all for vectors in the population and for all variables of each vector, decreases slowly from 1 at the beginning of the run ($t=0$) to 0. 1 as the number of generations t approaches T. Such decreases may assist the fine-tuning capabilities of the algorithm. In this approach, the value of given parameter changes according to a fully deterministic scheme. The user thus has full control of the parameter, and its value at a given time t is completely determined and predictable.

Second, it is possible to incorporate feedback from the search process, still using the same σ for all vectors in the population and for all variables of each vector. A well-known example of this type of parameter adaptation is Rechenberg's 1/5 Success Rule②, which states that the ratio of successful mutations to all mutations should be 1/5. Hence if the ratio is greater than 1/5 the step size should be increased, and if the ratio is less than 1/5 then it should be decreased. The rule is executed at periodic intervals, for instance, after k iterations each σ is reset by

$$\sigma' = \begin{cases} \sigma/c & p_s > 1/5 \\ \sigma c & p_s < 1/5 \\ \sigma & p_s = 1/5 \end{cases}$$

---

① D B Fogel, L Fogel, J W Atmar. Meta-evolutionary programming[C]//R R Chen. Proceedings of the 25th Asilomar Conference on Signals, Systems and Computers, 1991, pp. 540-545.

I Rechenberg. Evolution strategies: Optimierung Technisher Systeme nach Prinzipien des Biologishen Evolution[M]. Stuttgart: Frommann-Holzboog, 1973.

② A E Eiben, J E Smith. Introduction to Evolutionary Computation [M]. Berlin Heidelberg: Springer Verlag 2003, pp. 130-150.

where p_s is the relative frequency of successful mutations, measured over a number of trials, and the parameter c should be $0.817 \leqslant c \leqslant 1$①.

Using this mechanism, changes in the parameter values are now based on feedback from the search. The influence of the user on the parameter values is much less direct here than in the deterministic scheme above. Of course, the mechanism that embodies the link between the search process and parameter values is still a heuristic rule indicating how the changes should be made, but the values of $\sigma(t)$ are not deterministic (although they do come from a fixed set).

Third, it is possible to assign an individual mutation step size to each solution, that is, to extend the representation to individuals of length $n+1$ as $\langle x_1, x_2, \cdots, x_n, \sigma \rangle$, and apply some variation operators (e. g. Gaussian mutation and arithmetical crossover) to the values of x_i as well as to the σ value of an individual. In this way, not only the solution vector values (x_i) but also the mutation step size of an individual undergoes evolution. A possible solution introduced for evolution strategies② is:

$$\sigma' = \sigma \cdot e^{\tau \cdot N(0,1)} \tag{1-1}$$

$$x_i' = x_i + \sigma' \cdot N_i(0,1) \tag{1-2}$$

Observe that within this self-adaptive scheme the heuristic character of the mechanism resetting the parameter values is eliminated, and a certain value of σ acts on all values of a single individual.

If we change the granularity of the mutation step-size parameter and use a separate σ_i to each x_i, then we obtain an extended representation as $\langle x_1, x_2, \cdots, x_n, \sigma_1, \sigma_2, \cdots, \sigma_n \rangle$.

Then mutations can be performed by

$$\sigma' = \sigma \cdot e^{\tau \cdot N_i(0,1)}, \quad x_i' = x_i + \sigma_i' \cdot N_i(0,1)$$

This is a straightforward extension of Eq. (1-1) and Eq. (1-2), and indeed, very similar to Eq. (4-4) of *Introduction to Evolutionary Computation*③.

1.2.2 Changing the Penalty Coefficients

In the previous section we described different ways to modify a parameter

① H P Schwegel. Numerische Optimierung von Computer-Modellen Mittels der Evolutions strategie [J], Vol. 26 of ISR. Basel, Stuttgart: Birkhaeuser, 1977.

② A E Eiben, J E Smith. Introduction to Evolutionary Computation [M]. Berlin Heidelberg: Springer Verlag, 2003, pp. 130-150.

③ A E Eiben, J E Smith. Introduction to Evolutionary Computation [M]. Berlin Heidelberg: Springer Verlag, 2003, pp. 130-150.

controlling mutation. Several other components of an EA have natural parameters, and these parameters are traditionally tuned in one or other ways. Here we show that other components, such as the evaluation function (and consequently the fitness function) can also be parameterized and thus varied. While this is a less common option than tuning mutation (although it is practiced in the evolution of variable-length structures for parsimony pressure①), it may provide a useful mechanism for increasing the performance of an evolutionary algorithm.

When dealing with constrained optimization problems, penalty functions are often used②. A common technique is the method of static penalties③, which requires fixed user-supplied penalty parameters. The main reason for its widespread use is that it is the simplest technique to implement—it requires only the straightforward modification of the evaluation function as follows:

$$\text{eval}(\bar{x}) = f(\bar{x}) + W \cdot \text{penalty}(\bar{x})$$

where f is the objective function, and $\text{penalty}(\bar{x})$ is zero if no violation occurs, and is positive, otherwise. Usually, the penalty function is based on the distance of a solution from the feasible region, or on the effort to "repair" the solution, i. e. to force it into the feasible region. In many methods a set of functions $f_j (1 \leqslant j \leqslant m)$ is used to construct the penalty, where the function f_j measures the violation of the jth constraint in the following way.

$$f_j(\bar{x}) = \begin{cases} \max\{0, g_j(\bar{x})\} & 1 \leqslant j \leqslant q \\ |h_j(\bar{x})| & q+1 \leqslant j \leqslant m \end{cases} \tag{1-3}$$

W is a user-defined weight, prescribing how severely constraint violations are weighted. In the most traditional penalty approach the weight W does not change during the evolution process. We sketch three possible methods of changing the value of W.

First, we can replace the static parameter W by a dynamic parameter, e. g. a function $W(t)$. Just as for the mutation parameter σ, we can develop a heuristic that modifies the weight W over time. For example, in the method proposed by

---

① B Zhang, H Muhlenbeim. Balancing accuracy and parsimony in genetic programming [J]. Evolutionary Computing, 1995(3):3, pp. 17-38.

② A E Eiben, J E Smith. Introduction to Evolutionary Computation [M]. Berlin Heidelberg: Springer Verlag, 2003, pp. 130-150.

③ Z Mi Ichalewicz, M Schoenauer. Evolutionary algorithms for constrained parameter optimization problems[J]. Evolutionary Computation, 1996(4):1, pp. 1-32.