States based evolutionary algorithm

Sébastien Verel, Philippe Collard, Manuel Clergue

States based Evolutionary Algorithm

Sébastien Verel[1,2], Philippe Collard[2], and Manuel Clergue[2]

[1] INRIA Lille - Nord Europe, France
[2] University of Nice Sophia-Antipolis, France.

Abstract. Choosing the suitable representation, the operators and the values of the parameters of an evolutionary algorithm is one of the main problems to design an efficient algorithm for one particular optimization problem. This additional information to the evolutionary algorithm generally is called the algorithm parameter, or parameter. This work introduces a new evolutionary algorithm, *States based Evolutionary Algorithm* which is able to combine different evolutionary algorithms with different parameters included different representations in order to control the parameters and to take the advantage of each possible evolution algorithm during the optimization process. This paper gives first experimental arguments of the efficiency of the States based EA.

1 Introduction

The evolutionary algorithms are based on a search population of potential solutions of an optimisation problem. They iterate selection and replacement operators which favour the exploitation with stochastic operators which favour the exploration. Evolutionary Algorithm (EA) is a general framework which has been prove to be efficient on large optimisation classes of problems. The efficiency of the EA is lied in the ability to manage the balance between the exploitation and the exploration of the search space. The stochastic or selection operators must be chosen and their parameters must be tuned in order to be efficient on the given optimisation problem. And so, one main difficulty for an engineer or optimisation researcher is to choose such operator and parameters. More generally, the representation or coding of solutions either the fitness function is also a key in the design of EA [7,8]. The problem could be difficult for one representation or fitness function and not for another one [9]. Generally this additional information to the evolutionary algorithm is called *algorithm parameter*, or simply *parameter*; and the problem to choose efficient parameter *parameter setting*. Two ways of parameters setting are possible: first one is off-line before the run, often called *parameter tuning*, and the second is on-line during the run, called *parameter control*. A nice review of this field of research has been made in the recent work of Lobo *et al.* [5]. The algorithm proposed in this work will use a parameter control method instead of parameter tuning method. The parameter of the States based EA will be modified on-line during the run by taking into account the current state of the search.

2 States based Evolutionary Algorithm

To define the State based EA (SEA), we supposed to have n evolutionary algorithms. Each algorithm EA_i is a stochastic population based operator defined on the search spaces $\mathcal{S}$: $\forall i \in [1, n],\ EA_i : 2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}}$

In a practical way, each EA_i could have his own representation of solutions, but also his own selection operator, variation operators (mutation and crossover), and parameter settings. The EA has only to be defined whatever the population size.

A *state of a solution* is an integer number between $[1, n]$ added to the solution which indicates the EA applied on the solution. If different representations and fitness functions are used for each EA_i, we suppose that the fitness of a solution does not change when the state changes: $\forall i, j,\ f_i(s) = f_j(stateMut_{ij}(s))$ where $stateMut_{ij}$ is an operator which changes the state of solution s from i to j. The main principle of the SEA is to control the population size of each EA_i according to the fitness values of the actual solutions with a classical selection operator.

The total search population size is constant during the run and given by a parameter. The number of states n is also a parameter of the algorithm. The initialization of the population is an operator which first choose the state of each solution, and then apply the initialisation operator of the solutions state. One iteration of the algorithm is the succession of the stochastic population based operators: selection, split, EA_i, merge, state mutation, and replacement. The *operator of selection* choose a population from a population included in $\mathcal{S}$ of solutions with their state. It selects the solutions according to their fitness but not directly according to their state. The *splitting operator* splits the population from $\mathcal{S}$ into n sub-populations in $\mathcal{S}_i$. Each sub-population is composed with all the solutions in the same state. Then, the EA corresponding to the solutions state is applied on each sub-population. The EAs are independent, and does not take into account the result of the other EAs. The sizes of each sub-populations vary as they are not defined by a parameter of the algorithm but by the number of solutions in each state. This stage could be parallelized. The *merging operator* is a simple operator which merges the n sub-populations from $\mathcal{S}_i$ into one population of $\mathcal{S}$. After, a *operator of state mutation* is applied which possibly change only the state of solutions. If a solution changes from search space k to search space l, the solution is moved with the state change operator $stateMutation_{kl}$. Remember that the fitness of solutions is maintained by a state change. It is possible to have state mutation operators which changes the state of solutions in the same way for all solutions in the same states independently of the solutions in other state. In that case, it is possible to parallelize this stage. Otherwise, it is not possible when for example, the operator changes the state of a fixed number of solutions. The *operator of replacement* is the classical operator which creates a new population for the next iteration according to the population from EAs and the old population of solutions. Again, the replacement does not take into account the states of solutions. The algorithm 1 is the algorithm of State based EA.

The selection of the right state is indirect and it is made only with the fitness of the actual solutions. If the solutions of a given state are better, we hope that the selection operator choose more often such solutions, and as a consequence, the state population size should increase. Indirectly, the selection operator puts more solutions in the rights state.

Algorithm 1 States based evolutionary algorithm.

$population \leftarrow$ **initialisation**()
while continue($population$) **do**
  $selectedPop \leftarrow$ **selection**($population$)
  ($pop_1, \ldots, pop_n$) $\leftarrow$ **split**($selectedPop$)
  for all $i \in [1, n]$ **do**
    $pop_i \leftarrow$ **EA**$_i$(pop_i)
  end for
  $mergedPop \leftarrow$ **merge**($pop_1, \ldots, pop_n$)
  $child \leftarrow$ **stateMutation**($mergedPop$)
  $population \leftarrow$ **replace**($child$, $population$)
end while

3 Experimental study

To conduct experimental study of the SEA, we test the algorithm on two classical problems in EA: *OneMax* and *long k-path* problems. Those problems were used in recent works of Fialho et al. [1] [2] which propose a parameters control methods, the average and the extreme Dynamic Multi-Bandits (resp. avg-DMAB and ex-DMAB). It allows the comparison of performances between the SEA and the exDMAB. The authors takes $(1+\lambda)-$EA with $\lambda = 50$. To compare to this work, we use evolution algorithms without crossover, and using one of the same four mutation operators: the standard $1/l$ bit-flip operator (every bit is flipped with the binomial distribution of parameter $1/l$ where l is the length of the bit-strings), and the $1-$bit, $3-$bit, and $5-$bit mutation operators (the $b-$bit mutation flips exactly b bits, uniformly selected in the parents). So there are four evolutionary algorithms denoted EA_i $i = 1..4$. The population size of EA_i is λ_i. At each iteration of EA_i, the best solution is selected from the λ_i solutions in the parent population. From this best solution, λ_i solutions are created using the mutation operator. The generational replacement with elitism is used: all the offspring solutions replace the parent ones keeping the best solution founded in the parent population. Those EA_i are closed to $(1 + \lambda)-$EA.

The SEA controls the population size of the four EA_i. The total population size is $\lambda = \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 50$. The selection method used is tournament selection of size t, and the replacement method is generational with elitism. The state mutation operator changes a state at random with a rate p. Each EA_i is running with the number of generation g. The meta-parameters of the SEA has been computed off-line by a Design Of Experiments campaign. The tournament selection is used for the state selection operator with the tournament size $t \in \{2, 3, 5\}$. The mutation operator changes the state of solutions randomly with the state mutation rate $p \in \{0.001, 0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.4\}$. The number of generations of each EA_i is $g \in \{1, 5, 10, 25, 50, 100, 200, 400\}$.

3.1 One Max problem

The oneMax problem, the ”drosophila” of evolutionary computation, is a unimodal problem defined on binary strings of size l. The fitness is the number of ”1” in the

bit-string. In the experiments, the length of the bit strings is $l = 10^4$. In all reported experiments, the initial solution is set to $(0, \ldots, 0)$.

The table 1 shows the performances of the SEA compared to the ones of extreme and average DMAB (ex-DMAB and avg-DMAB) and oracle strategy reported from [1]. The performance of SEA are better than the average-DMAB and worst than the extreme-DMAB according to the non-parametric tests of Wilcoxon and Kolmogorov-Smirnov (p-values are under 10^{-7}). The figure 3 shows a typical example of dynamics of the SEA. Due to the difference of the number of iteration between the 200 runs, we show the run which gives the median performance (5891 generations). At the beginning, the largest sub-population quickly becomes the one of the 5-bit mutation operator. This operator is the best one at this stage. Around the generation 3600, the population size of the 1-bit mutation operator dominates the other ones. We can notice that the 3-bit mutation operator never able to dominate the others. The 3-bit mutation is the operator with best average fitness gain for fitness values between 6580 and 8400. Nevertheless, the fitness gain of the 5 and 3 bit mutations are close. The difference of performance between SEA and ex-DMAB could be explain by the relative lack of the 3 bit mutation operator. At the end, the $1/l$ bit-flip mutation has a population size around those of the 1-bit mutation. Indeed, this two operators have nearly the fitness gain for high fitness value.

Table 1. Average and standard deviation number of generations to the optimal solution out of 200 independent (for SEA) runs using the optimal meta-parameters

Algo.	Configuration	Gens to Opt. (avg_{sd})
SEA	$t = 2\ p = 0.05\ g = 25$	6076_{804}
avg-DMAB	$C = 10\ \gamma = 25\ W = 50$	7727_{642}
Ex-DMAB	$C = 1\ \gamma = 250\ W = 50$	5467_{513}
Oracle strategy		5069_{292}

The figure 2 allows to study the robustness of the meta-parameters of the SEA. It shows the average performances of the SEA computed off-line during the Design Of Experiments campaign according to the tournament size t, the state mutation rate p, and the number of generations g. The average performance is relatively independent of the tournament size. The number of generations and the state mutation rate have more impact on it. Extreme values of the number of generations (1, 200, and 400) do not give good performances. According to the state mutation rate, the performances increases till p around 0.05 or 0.1, and decreases after 0.1. For the oneMax problem with $l = 10^4$ robust parameters seems to be $t \in [2, 5]$, $p \in [0.05, 0.1]$, and $g \in [25, 50]$. In this case, the window of robust meta-parameters is quite large.

3.2 Long k-path problem

The long path problem [4] has been introduced to show that a problem instance can be difficult to solve for a hill-climber-like heuristic even if the search space is unimodal,

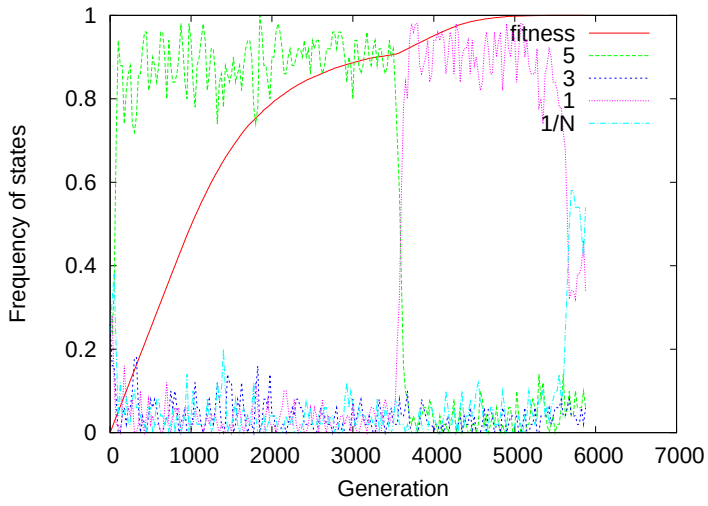

Fig. 1. Example of run of the SEA with $t = 2$, $p = 0.05$ and $g = 25$ on the oneMax problem: fitness, the relative number of solutions in each states are given. The chosen example gives the median performance (gens to opt 5891).

i.e. a fitness landscape where the global optima is the single local optima. For such a problem, a hill-climber guarantees to reach the global optimum, but the length of the path to get it is exponential in the dimension of the search space. As a consequence, a hill-climbing-based heuristic cannot solve the problem in polynomial time. The 'path length' takes then place in the rank of problem difficulty, on the same level as multi-modality, ruggedness, deceptivity, etc. The long path problem [10] [3] can be solved in a polynomial expected amount of time for a $(1+1)-$EA which is able to mutate more than one bit at a time. This $(1+1)-$EA can take some shortcuts on the outside of the path so that it makes the computation more efficient.

Like in the work of Fialho, an additional mutation operator, the $3/l$ bit-flip, has been added to the operator set. The table 2 shows the performances of the SEA and ex-DMAB. The minimum and the median of number generations to optimum over 200 runs are presented. The performances for all algorithms are highly variables from one sample to another. We perform the two non-parametric tests of Wilcoxon (W) and Kolmogorov-Smirnov (KS). For $l = 43$, the performances of ex-DMAB are better according to KS with confidence 1%, but only at confidence 5% for W test. For $l = 49$, the two algorithms obtain similar results according both test W and KS. For the problems with $l = 55$ and $l = 61$, the SEA outperforms the ex-DMAB according the statistical tests (p-values are under 10^{-4}). It is difficult to have a clear conclusion from those experiments due to the large randomness of the results, but it seems that SEA is able to have better performances than extreme value based DMAB on the largest instance of long 3-path problems.

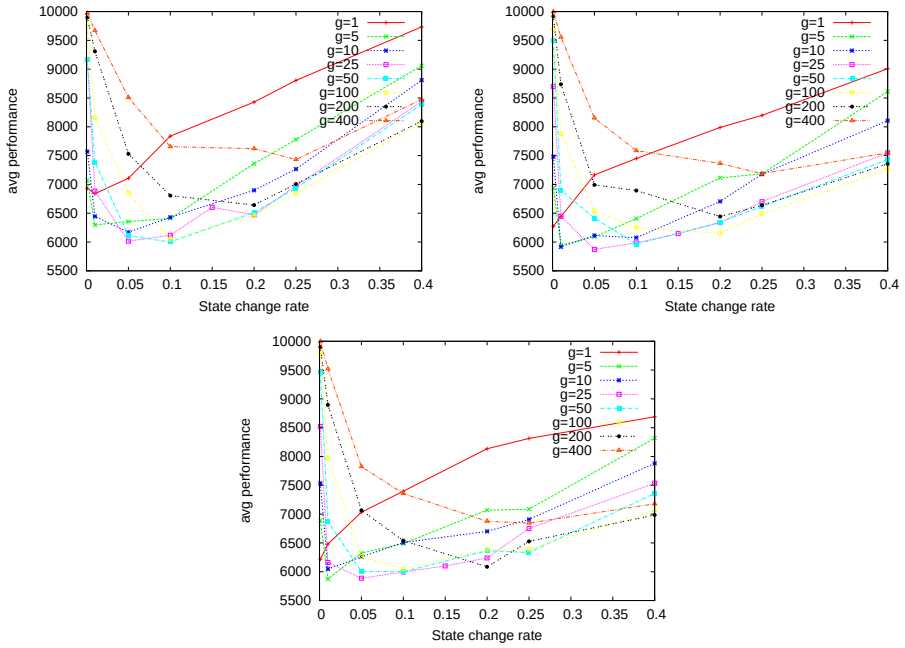

Fig. 2. Average performance of the SEA on the oneMax problem with $l = 10^4$ computed off-line during the Design Of Experiments campaign according to the state mutation rate p, the number of generation g, and tournament size t. From top to bottom tournament size $t = 2, 3, 5$. Average on 50 independent runs.

Table 2. Minimum and median number of generations to the optimal solution out of 200 independent runs using the optimal meta-parameters

l	Algo.	Configuration	Gens to Opt. $(min - median)$
43	SEA	$t = 2\ p = 0.15\ g = 10$	$20 - 3151$
	Ex-DMAB	$C = 50\ \gamma = 50\ W = 500$	$11 - 2216$
	Oracle strategy		$2 - 1202$
49	SEA	$t = 2\ p = 0.4\ g = 5$	$5 - 3755$
	Ex-DMAB	$C = 100\ \gamma = 500\ W = 500$	$17 - 3244$
	Oracle strategy		$19 - 2668$
55	SEA	$t = 2\ p = 0.1\ g = 5$	$51 - 4126$
	Ex-DMAB	$C = 100\ \gamma = 100\ W = 500$	$161 - 6190$
	Oracle strategy		$45 - 3224$
61	SEA	$t = 2\ p = 0.1\ g = 50$	$59 - 7950$
	Ex-DMAB	$C = 50\ \gamma = 25\ W = 500$	$80 - 10253$
	Oracle strategy		$8 - 5408$

The figure 3 displays a typical example of run of the SEA on the long 3-path with $l = 55$. Again, we choose to show the one with the median performance (4126 generations). The search dynamics on long path problems is different from the one on the oneMax problem. There is no large stage where the number of solution in one state dominates the others. Even the larger sub-population size variation is not regular, and draws a lot of random jumps. Those stochastic variations of sub-population size seem to appear all along the search process, there is no difference between the beginning and the end of the run. We can notice that the sub-population of the $1/l$ bit-flip mutation operator is more often the larger one. For this typical run, the $1/l$ bit-flip mutation is the larger one during 60% of the total number of generation, the 1-bit mutation 21% of times, the 3-bit mutation 12%, the $3/l$ bit-flip mutation 5%, and the less used is the 5-bit mutation with 1% of the total number of generations. For the most used mutation operator, the sub-population size is around 55% of the total number of solutions λ. It also means that the other mutations are always applied during the run.

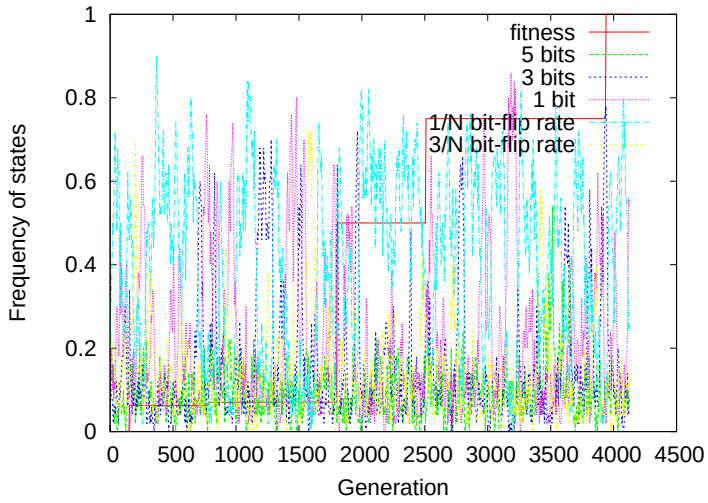

Fig. 3. Example of run of the SEA with $t = 2$, $p = 0.1$ and $g = 5$ on the long 3-path problem with $l = 55$: fitness, the relative number of solutions in each states are given. The chosen example gives the median performance (gens to opt 4126).

The robustness of the meta-parameters of the SEA is also study for the long path problems. The figure 2 shows the median generation to reach the global optimum computed off-line during the Design Of Experiments campaign according to the state mutation rate p, and the number of generations g. Only the tournament size $t = 2$ is shown. The similar performances are obtained for tournament size 3 and 5. A larger number of generation for the EA_i gives bad performances, and g must be lower than 100. For this problem $g = 1$ can be used in contrary of the oneMax problem. The state mutation

rate seems to be more robust for this problem. Even if a rate around 0.1 improves the performances of the SEA. The large set of parameters values gives good performances. The meta-parameters of the SEA are more robust for the long path problems than for the oneMax problem.

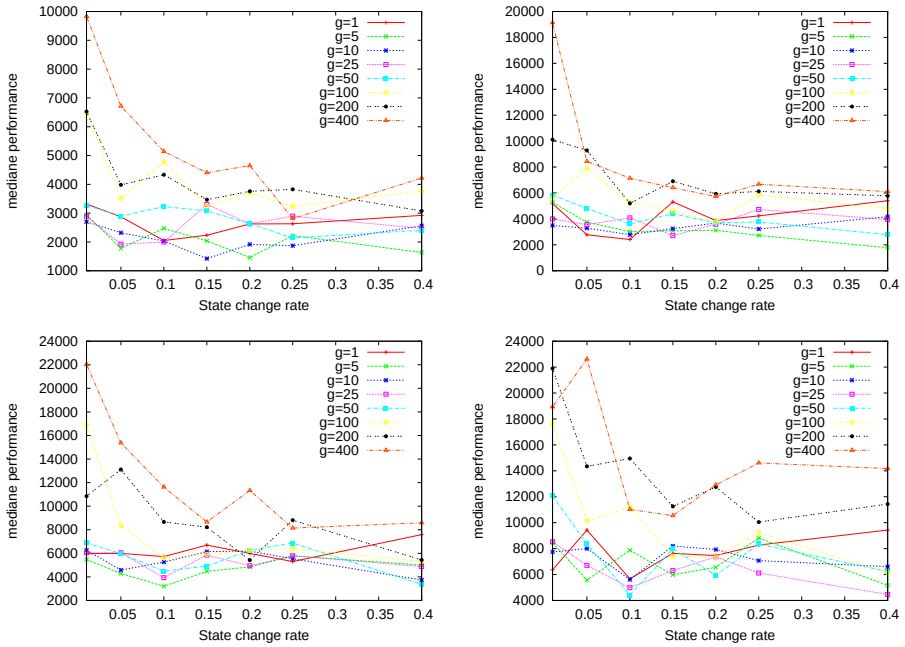

Fig. 4. Median generation (over 50 independent runs) to optimal solution on the long 3-path problems computed off-line during the Design Of Experiments campaign according to the state mutation rate. Tournament size is $t = 2$. From top-left to bottom-right $l = 43, 49, 55, 61$.

4 Discussion

In this paper we have presented the State based EA (SEA) as a general framework which allows to choose efficient parameter setting on-line during the run. Most often, such approaches work with measure of performance on the operator used as evolvability of solution or diversity of the population. The SEA proposes a new method which uses directly the distribution of fitness of solutions produced by the operator to implicitly select the parameters. Nevertheless, it is rather an adaptive method than an auto-adaptive method because it manages two kind of dynamics: one to evolve the solutions, and one to evolve the parameters of the first ones. The rate of evolution of EA parameters

is different than those of the meta-parameters. A crucial point is to well adjust the two dynamics in order they work in concordance. Like the bandit methods used in previous works, the SEA uses an exploitation component (the global selection) and an exploration component (the state mutation). The way to manage the computational cost, *i.e.* the number of evaluations, between the operators (or evolutionary algorithms) is different. In the AOS methods, one operator (or optimization algorithm) is used at each generation, and the method controls the number of generation for each operator during the run. In the SEA, several evolutionary algorithms can be used at the same time, and the method controls the sub-population size of each algorithm. The SEA is more parallel and the AOS more sequential.

The performance of the SEA should clearly depends on the balance between the exploitation and the exploration components. So, theoretical works must be done to analyse and coordinate this balance. New experimental studies must be conducted on others problems to understand the advantageous and the limits the SEA. The oneMax and long path problems are unimodal problems where the population based algorithms does not perform better than algorithms like $(1 + \lambda)$-ES. So, next step will be study the performance of the SEA on problems such as SAT problems, and then compare the SEA with the extreme Compass Dynamic Multi-Armed Bandit [6].

Acknowledgments

The authors would thank Alvaro Fialho for his grateful help.

References

1. Álvaro Fialho, Luis Da Costa, Marc Schoenauer, and Michèle Sebag. Dynamic Multi-Armed Bandits and Extreme Value-based Rewards for Adaptive Operator Selection in Evolutionary Algorithms. In *Learning and Intelligent Optimization (LION 3)*, Trento Italie, 2009. BEST PAPER AWARD I.: Computing Methodologies/I.2: ARTIFICIAL INTELLIGENCE/I.2.8: Problem Solving, Control Methods, and Search.
2. Álvaro Fialho, Marc Schoenauer, and Michèle Sebag. Analysis of adaptive operator selection techniques on the royal road and long k-path problems. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 779–786, New York, NY, USA, 2009. ACM.
3. Josselin Garnier and Leila Kallel. Statistical distribution of the convergence time of evolutionary algorithms for long-path problems. *IEEE Trans. Evolutionary Computation*, 4(1):16–30, 2000.
4. Jeffrey Horn, David E. Goldberg, and Kalyanmoy Deb. Long path problems. In *PPSN*, volume 866 of *LNCS*, pages 149–158. Springer, 1994.
5. Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.
6. Jorge Maturana, Álvaro Fialho, Frederic Saubion, Marc Schoenauer, and Michele Sebag. Extreme compass and dynamic multi-armed bandits for adaptive operator selection. In *CEC'09: Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 365–372. IEEE Press, May 2009.

7. Franz Rothlauf and David E. Goldberg. *Representations for Genetic and Evolutionary Algorithms*. Physica-Verlag, 2002.
8. Franz Rothlauf and David E. Goldberg. Redundant representations in evolutionary computation. *Evol. Comput.*, 11(4):381–415, 2003.
9. Jonathan Rowe, Darrell Whitley, Laura Barbulescu, and Jean-Paul Watson. Properties of gray and binary representations. *Evol. Comput.*, 12(1):47–76, 2004.
10. Günter Rudolph. How mutation and selection solve long path problems in polynomial expected time. *Evolutionary Computation*, 4(2):195–205, 1996.