2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

Boosting Genetic Algorithms with Self-Adaptive Selection

A.E. Eiben, M.C. Schut and A.R. de Wilde

***Abstract*— In this paper we evaluate a new approach to selection in Genetic Algorithms (GAs). The basis of our approach is that the selection pressure is not a superimposed parameter defined by the user or some Boltzmann mechanism. Rather, it is an *aggregated parameter* that is determined collectively by the individuals in the population. We implement this idea in two different ways and experimentally evaluate the resulting genetic algorithms on a range of fitness landscapes. We observe that this new style of selection can lead to 30-40% performance increase in terms of speed.**

I. Introduction

Parameter control is a long-standing grand challenge in evolutionary computing. The motivation behind this interest is mainly twofold: performance increase and the vision of parameterless evolutionary algorithms (EA). The traditional mainstream of research concentrated on adaptive or self-adaptive control of the variation operators, mutation and recombination [15], [9], [11]. However, there is recent evidence, or at least strong indication, that "tweaking" other EA components can be more rewarding. In [2], [7] we have addressed population size, in the present study we consider selection.

Our approach to selection is based on a radically new philosophy. Before presenting the details, let us first make a few observations about (self-)adaptation and the control of selection parameters. Globally, two major forms of setting parameter values in EAs are distinguished: *parameter tuning* and *parameter control* [6], [8]. By parameter tuning we mean the commonly practiced approach that amounts to 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. Parameter control forms an alternative, as it amounts to starting a run with initial parameter values that are changed *during* the run. Further distinction is made based on the manner changing the value of a parameter is realized (i.e., the "how-aspect"). Parameter control mechanisms can be classified into one of the following three categories.

- **Deterministic parameter control** This takes place when the value of a strategy parameter is altered by some deterministic rule modifying the strategy parameter in a fixed, predetermined (i.e., user-specified) way without using any feedback from the search. Usually, a time-dependent schedule is used.
- **Adaptive parameter control** This takes place when there is some form of feedback from the search that serves as inputs to a mechanism used to determine the change to the strategy parameter. The assignment of the value of the strategy parameter, often determined by an IF-THEN rule, may involve credit assignment, based on the quality of solutions discovered by different operators/parameters, so that the updating mechanism can distinguish between the merits of competing strategies. Although the subsequent action of the EA may determine whether or not the new value persists or propagates throughout the population, the important point to note is that the updating mechanism used to control parameter values is externally supplied, rather than being part of the "standard" evolutionary cycle.
- **Self-adaptive parameter control** Here the parameters to be controlled are encoded into the chromosomes and undergo mutation and recombination. The better values of these encoded parameters lead to better individuals, which in turn are more likely to survive and produce offspring and hence propagate these better parameter values. This is an important distinction between adaptive and self-adaptive schemes: in the latter the mechanisms for the credit assignment and updating of different strategy parameters are entirely implicit, i.e., they are the selection and variation operators of the evolutionary cycle itself.

In terms of these categories it can be observed that parameters regarding selection and population issues (e.g., tournament size or population size) are of global nature. They concern the whole population and cannot be naturally decomposed and made local, i.e., cannot be defined at the level of individuals, like mutation step size in evolution strategies. Consequently, existing approaches to controlling such parameters are either deterministic or adaptive. Self-adaptive population and selection parameters do not seem to make sense. This, is however, exactly what we are doing here.

The basis of our approach is that –even if it sounds infeasible– we do make the selection pressure locally defined, at individual level. Thus it will not be a global level parameter defined by the user or some time-varying Boltzmann mechanism. Rather, it is determined collectively by the individuals in the population via an aggregation mechanism. On pure theoretical grounds it is hard to predict whether this approach would work. On the one hand, individuals might be interested in "voting" for low selection pressure

A.E. Eiben, M.C. Schut and A.R. de Wilde are with the Department of Computer Science, Vrije Universiteit, Amsterdam (email: {AE.Eiben, MC.Schut}@few.vu.nl).

thereby frustrating the whole evolutionary process. Such an effect occurs in the well-known tragedy-of-the-commons: "self-maximizing gains by individuals ultimately destroy the resource, such that nobody wins" [13]. On the other hand, self-adaptation is generally acknowledged as a powerful mechanism for regulating EA parameters. So after all, an experimental study seems to be the only practical way to assess the value of this idea – which is what this paper delivers.

Our technical solution is based on assigning an extra parameter to each individual representing the individual's "vote" in the collective decision regarding the tournament size. Depending on how these parameters are updated during the search (a run of the EA), we obtain two types of methods.

- If the update takes place through a feedback rule of the form
  IF *condition* THEN *newvalue-1* ELSE *newvalue-2*
  then we have an adaptive selection parameter.
- If we postulate that the new parameter is just another gene in the individuals that undergoes mutation and recombination with the other genes, then we have a self-adaptive selection parameter.

To be concrete, the research questions to be answered here are the following:

1) Does self-adaptive selection, based on the "voting" idea work? That is, does an EA augmented with such selection outperform a regular EA?
2) If it does, does it work better than adaptive selection? That is, using an (almost) identical mechanism in adaptive and self-adaptive fashion, which option yields a better EA?

The paper is structured as follows. In Section II we describe the details of a self-adaptive and an adaptive mechanism for controlling the tournament size in a GA. (NB. As we will discuss later, it can be argued that our adaptive mechanism is a combination of a heuristic and self-adaptivity, but we will be pragmatic about this terminology issue.) In Section III we present the test suite used in our experiments, the performance measures monitored for comparing different GAs, and the specifications of the benchmark GA and the (self-)adaptive GAs. The experimental results are given in Section IV. Finally, we round up with drawing conclusions and pointing to further research.

II. Mechanisms for (self-)adapting tournament size

While the picture regarding the control of variation operators in EAs is rather diverse, most researchers agree that increasing the selection pressure as the evolutionary process goes on offers advantages [1], [18], [5], [16], [10], [14], [12]. In the present investigation we will introduce as little as possible bias towards increasing or decreasing the parameter values and make selection pressure an observable.

The basis for both mechanisms is an extra parameter $k \in [k_{min}, k_{max}]$ in each individual and a "voting" mechanism that determines the tournament size K used in the GA. It is important to note that tournament size K is a parameter that is valid for the whole population, i.e., for all selection acts. Roughly speaking, the tournament size K will be the sum of selection size parameters of all individuals k_i calculated as follows:

$$K = \lceil \sum_{i=1}^{N} k_i \rceil \tag{1}$$

where $k_i \in (0,1)$ are uniform randomly initialised, $\lceil \; \rceil$ denotes the ceiling function, $K \in [1, N]$, and N is the population size.

A. Self-adaptation of tournament size

The self-adaptive scheme is conceptually very simple. Technically, our solution is twofold:

1) We postulate that the extra parameter k is part of the individual's chromosomes, i.e., an extra gene. The individuals are therefore of the form $\langle x, k \rangle$, where x is the bitstring and k is the parameter value.
2) Declare that crossover will work on the whole (extended) chromosome, but split mutation into two operations. For the x part we keep the regular GA mutation (whichever version is used in a given case), but to mutate the extra genes we define a specific mechanism.

Finding an appropriate way to mutate the k-values needs some care. A straightforward option would be the standard self-adaptation mechanism of σ values from Evolution Strategies. However, those σ values are not bounded, while in our case $k \in [k_{min}, k_{max}]$ must hold. We found a solution in the self-adaptive mechanism for mutation rates in GAs as described by Bäck and Schütz [4]. This mechanism is introduced for $p \in (0,1)$ and it works as follows:

$$p' = \left(1 + \frac{1-p}{p} \cdot e^{-\gamma \cdot N(0,1)}\right)^{-1} \tag{2}$$

where p is the parameter in question, $N(0,1)$ is a normal distribution with mean 0 and standard deviation 1, and γ is the learning rate which allows for control of the adaptation speed. This mechanism has some desirable properties:

1) Changing $p \in (0,1)$ yields a $p' \in (0,1)$.
2) Small changes are more likely than large ones.
3) The expected change of p by repeatedly changing it equals zero (which is desirable, because natural selection should be the only force bringing a direction in the evolution process).
4) Modifying by a factor c occurs with the same probability as a modification by $1/c$.

In the present study, we use tournament size parameters $k \in (0,1)$ and the straightforward formula

$$k' = \left(1 + \frac{1-k}{k} \cdot e^{-\gamma \cdot N(0,1)}\right)^{-1}, \tag{3}$$

where $\gamma = 0.22$ (value recommended in [4]).

Let us note that if a GA uses a recombination operator then this operator will be applied to the tournament size parameter k, just as it is applied to to other genes. In practice this means that a child created by recombination inherits an initial k value from its parents and the definitive value k' is obtained by mutation as described by Equation 3.

B. Hybrid self-adaptation of tournament size

In the self-adaptive algorithm as described above the direction (+ or –) as well as the extent (increment/decrement) of the change are fully determined by the random scheme. This is a general property of self-adaptation. However, in the particular case of regulating selection pressure we do have some intuition about the direction of change. Namely, if a new individual is better than its parents then it should try to increase selection pressure, assuming that stronger selection will be advantageous for him, giving a reproductive advantage over less fit individuals. In the opposite case, if it is less fit than its parents, then it should try to lower the selection pressure. Our second mechanism is based on this idea. Formally, we keep the aggregation mechanism from equation 1 and use the following rule. If $\langle x, k \rangle$ is an individual to be mutated (either obtained by crossover or just to be reproduced solely by mutation), then first we create $\mathbf{x}'$ from $\mathbf{x}$ by the regular bitflips, then apply

$$k' = \begin{cases} k + \Delta k & if \quad f(\mathbf{x}') \geq f(\mathbf{x}) \\ k - \Delta k & otherwise \end{cases} \tag{4}$$

where

$$\Delta k = \left| k - \left(1 + \frac{1-k}{k} e^{-\gamma N(0,1)} \right)^{-1} \right| \tag{5}$$

with $\gamma = 0.22$.

This mechanism differs from “pure” self-adaptation because of the heuristic rule specifying the direction of the change. However, it could be argued that this mechanism is not a clean adaptive scheme (because the initial k values are inherited), nor a clean self-adaptive scheme (because the final k values are influenced by a user defined heuristic), but some hybrid form. For this reason we perceive and name this mechanism *hybrid self-adaptive* (HSA).

III. Experimental setup

A. Test suite

The test suite[1] for testing GAs is obtained through the Multimodal Problem Generator of Spears [17]. We generate landscapes of 1, 2, 5, 10, 25, 50, 100, 250, 500 and 1000 binary peaks $\mathcal{P}$ whose heights are linearly distributed and where the lowest peak is 0.5. The length L of these bit strings is 100. The fitness of an individual is measured by the Hamming distance between the individual and the nearest peak, scaled by the height of that peak. The nearest peak is determined by

$$Peak_{near}(\mathbf{x}) = \min_{i=1}^{\mathcal{P}} (Hamming(\mathbf{x}, Peak_i)),$$

in case of multiple peaks at the same distance, the highest neighboring peak is chosen. Then, the evaluation function of an individual is $f(\mathbf{x}) =$

$$\frac{L - Hamming(\mathbf{x}, Peak_{near}(\mathbf{x}))}{L} height(Peak_{near}(\mathbf{x})).$$

Note that the global maximum of $f(\mathbf{x}) = 1$.

B. Algorithm setup

GA model	steady-state
Representation	100-bitstring
Recombination	uniform crossover
Recombination probability	0.5
Mutation	bit-flip
Mutation probability	0.01
Parent selection	k-tournament
Survival selection	delete-worst-two
Population size	100
Initialisation	random
Termination	$f(x) = 1$ or 10,000 evaluations

TABLE I
DESCRIPTION OF THE SGA.

We compare three GAs in our experimental study: the Simple GA (SGA) as benchmark, the GA with hybrid self-adaptive tournament size (GAHSAT), and the GA with self-adaptive tournament size (GASAT). For all of these GAs the following holds. In pseudo code the algorithm works as follows:

```
begin
   INITIALIZE population with random individuals
   EVALUATE each individual
   while not stop-condition do
      SELECT two parents from the population
      RECOMBINE the parents
      MUTATE both resulting children
      REPLACE the two worst individuals by both children
   od
end
```

The parameters of the SGA are shown in I; GAHSAT and GASAT have similar setting with obvious differences (representation, mutation, crossover) as described above. The chromosome of each individual consists of $L = 100$ binary genes and one real value for $k \in (0, 1)$. It is a steady state GA with delete worst two replacement. The static independent variables are the uniform mutation probability $p_m = 1/L = 0.01$, the uniform crossover probability $p_c = 0.5$, the population size $N = L = 100$. The GA terminates if the optimum of $f(\mathbf{x})$, what is equal to one, is reached or when 10,000 individuals are evaluated.

Obviously, the GAs differ in their selection mechanisms:

- SGA works with tournament size $K = 2$,
- GASAT works with the mechanism described in Section II-A,
- GAHSAT works with the mechanism described in Section II-B.

[1] The test suite can be obtained from the webpage of the authors.

C. Performance measures

During running a GA, each generation is monitored by measuring the Best Fitness (BF), Mean Fitness (MF), Worst Fitness (WF) and the diversity of the population. After 100 runs, data of the monitors is collected and the Mean Best Fitness (MBF) and its standard deviation (SDMBF), the Average number of Evaluations to a Solution (AES) and its standard deviation (SDAES) and the Success Rate (SR) will be calculated.

IV. RESULTS

SGA

In table II is to see that the harder the problem, the lower the SR and MBF and the higher the AES.

GAHSAT

In table III the results for GAHSAT are given. These results are promising because the AES values are much lower than those of the SGA. This indicates that on-the-fly adjustment of K contributes to a faster GA. Comparing the SRs and MBF results (and the SD's) of SGA and GAHSAT makes clear that the extra speed of GAHSAT comes at no extra costs in terms of solution quality or stability.

GASAT

The outcomes for GASAT in Table IV indicate that the "purely" self-adaptive selection is not as powerful as the adaptive or hybrid self-adaptive mechanism (in table III). But nevertheless GASAT has better performance than SGA, reconfirming that on-the-fly adjustment of K is beneficial.

Peaks	SR	AES	SDAES	MBF	SDMBF
1	100	1478	191	1.0	0.0
2	100	1454	143	1.0	0.0
5	100	1488	159	1.0	0.0
10	93	1529	168	0.9961	0.0142
25	62	1674	238	0.9885	0.0174
50	37	1668	221	0.9876	0.0140
100	22	1822	198	0.9853	0.0145
250	11	1923	206	0.9847	0.0137
500	6	2089	230	0.9865	0.0122
1000	5	2358	398	0.9891	0.0100

TABLE II

END RESULTS OF SGA.

Looking at the outcomes from the perspective of algorithm performance it is clear that the three GAs do not differ much in terms of MBF and SR. Apparently, the maximum number of fitness evaluations is large enough to get the same solution quality with all three setups. However, the GAs do differ in speed, expressed in terms of the Average number of Evaluations to a Solution. Simple arithmetic comparison indicates 30-40% speed increase with respect to the SGA.

To check if GAHSAT was indeed the overall winner, we performed several statistical tests. Here we only present the

Peaks	SR	AES	SDAES	MBF	SDMBF
1	100	989	244	1.0	0.0
2	100	969	206	1.0	0.0
5	100	1007	233	1.0	0.0
10	89	1075	280	0.9939	0.0175
25	63	1134	303	0.9879	0.0190
50	45	1194	215	0.9891	0.0127
100	14	1263	220	0.9847	0.0140
250	12	1217	166	0.9850	0.0131
500	7	1541	446	0.9876	0.0119
1000	4	1503	272	0.9862	0.0113

TABLE III

END RESULTS OF GAHSAT.

Peaks	SR	AES	SDAES	MBF	SDMBF
1	100	1312	218	1.0	0.0
2	100	1350	214	1.0	0.0
5	100	1351	254	1.0	0.0
10	92	1433	248	0.9956	0.0151
25	62	1485	280	0.9893	0.0164
50	46	1557	246	0.9897	0.0128
100	21	1669	347	0.9853	0.0147
250	16	1635	336	0.9867	0.0130
500	3	1918	352	0.9834	0.0146
1000	1	1675	0	0.9838	0.0126

TABLE IV

END RESULTS OF GASAT.

t-test data for testing the following three hypotheses:

$$\begin{aligned} H_0 &: \bar{x}_1 = \bar{x}_2, \\ H_{a_1} &: \bar{x}_1 \neq \bar{x}_2, \\ H_{a_2} &: \bar{x}_1 > \bar{x}_2, \end{aligned}$$

where $\bar{x_1}$ stands for the AES of GAHSAT and $\bar{x_2}$ stands for the AES of GASAT. The results of these statistical analysis are shown in Table V. In the second column the p-value of H_0 against H_{a_1} is given. The fourth column shows the p-value of H_0 against H_{a_2}. And the third column shows the conclusion drawn by a 5%-significance level for each of the ten problem instances. The conclusions of the ten p-values in the third column are taken into account to determine if there is a significant difference. This statistical analysis accompanied with the Success Rate (SR) and Mean Best Fitness (MBF) of the given GAs confirm that GAHSAT outperforms GASAT that in turn outperforms SGA.

Additionally to recording and comparing performance related measures we also looked into algorithm behavior. In particular, to gain some insight into the workings of these mechanisms for a number of runs we have plotted

- the best/mean/worst fitness of the population, together with the population diversity measured by entropy;
- the development of K during a run.

The first type of plots in Figure 2 disclose clear differences between the three GAs. Compared to the benchmark SGA the GASAT variant shows faster decrease of diversity and a faster increase of solution quality (fitness). The curves belonging

Peaks	p		p'
1	0.00	$<$	1.00
2	0.00	$<$	1.00
5	0.00	$<$	1.00
10	0.00	$<$	1.00
25	0.00	$<$	1.00
50	0.00	$<$	1.00
100	0.00	$<$	1.00
250	0.00	$<$	1.00
500	0.23	$=$	0.88
1000	0.61	$=$	0.69

TABLE V

t-TEST RESULTS FOR THE FOUND RANKING.

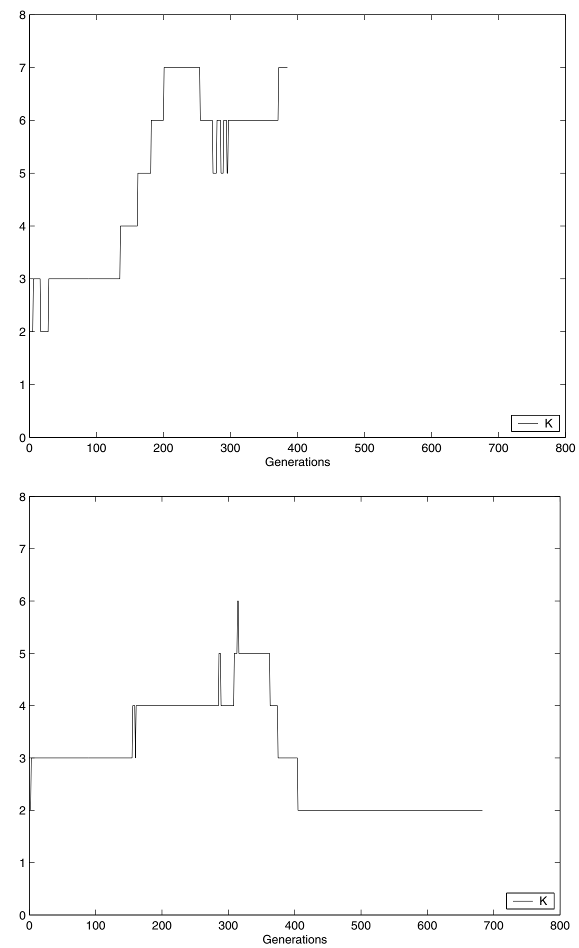

Fig. 1. Development of K for GAHSAT (top) and GASAT (bottom).

to GAHSAT show the same effect, but even stronger. The reason for this algorithm behavior can be found in the curves of Figure 1 that exhibit the development of K during a run. As shown in those plots the selection pressure is increasing (until approximately generation 300). In the large, this implies that formula 4 results in increasing K. However, while it is theoretically possible (and numerically reasonable to assume) that K would continue to grow, Figure 1 shows that this does not happen, but for GASAT it stabilizes at $K = 2$.

V. CONCLUSIONS

In this paper we have presented experimental evidence that varying the selection pressure on-the-fly can significantly improve the performance of a GA. This is, perhaps, not such a novel result per se. However, it is interesting that here we use a simple mechanism[2] and apply no sophisticated twists to it. Yet we obtain a GA that compares favorably with the best GA we found for the same test suite in an earlier paper [7]. The comparison between the former winner, a GA with adaptive population size (APGA), and GAHSAT is shown in Table VI. Note that the MBF results are omitted for they showed no significant difference. This comparison shows that the GAHSAT is very competitive, running out the APGA on the smoother landscapes.

Note also that the mechanism includes an additional parameter (the learning rate). From the perspective of increasing EA performance this is not a problem. However, from the parameterless evolutionary algorithms angle this may be considered problematic, because we eliminate one parameter (K) at the cost of introducing another one (γ). Nevertheless, in general, we believe that meta-parameters, i.e., those of a learning method extending the EA, are less sensitive for an accurate setting that the technical parameters, i.e., those of the EA itself. Furthermore, in our particular case we did not tune γ but used the recommended value and found that it worked well.

	GAHSAT		APGA	
Peaks	SR	AES	SR	AES
1	100	989	100	1100
2	100	969	100	1129
5	100	1007	100	1119
10	89	1075	95	1104
25	63	1134	54	1122
50	45	1194	35	1153
100	14	1263	22	1216
250	12	1217	8	1040
500	7	1541	6	1161
1000	4	1503	1	910

TABLE VI

COMPARING GAHSAT AND THE WINNING APGA FROM [7].

As for the research questions from the introduction, we have demonstrated that self-adapting a global parameter, like the tournament size K, is possible and can lead to better GA performance. This is a new result. Regarding the second research question, we found that regulating K by pure inheritance is not as powerful as regulating it by a heuristic *and* inheritance. This work is being followed up by applying the idea of aggregating local votes for other global parameters, like population size. Furthermore, we are to make a broad comparative study on the effect of the present parameter update mechanism (formula 2) on other parameters.

[2]that was, nota bene, introduced for mutation parameters

REFERENCES

[1] E.H.L. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines*. Wiley, Chichester, UK, 1989.

[2] T. Bäck, A.E. Eiben, and N.A.L. van der Vaart. An empirical study on GAs "without parameters". In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, and H.-P. Schwefel, editors, *Proceedings of the 6th Conference on Parallel Problem Solving from Nature*, number 1917 in Lecture Notes in Computer Science, pages 315–324. Springer, Berlin, 2000.

[3] T. Bäck, D.B. Fogel, and Z. Michalewicz, editors. *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, Bristol, 2000.

[4] Th. Bäck and M. Schütz. Intelligent mutation rate control in canonical genetic algorithms. In Zbigniew W. Ras and Maciej Michalewicz, editors, *Foundations of Intelligent Systems, 9th International Symposium, ISMIS '96, Zakopane, Poland, June 9-13, 1996, Proceedings*, volume 1079 of *Lecture Notes in Computer Science*, pages 158–167. Springer, Berlin, Heidelberg, New York, 1996.

[5] A. Dukkipati, N. M. Musti, and S. Bhatnagar. Cauchy annealing schedule: An annealing schedule for boltzmann selection scheme in evolutionary algorithms. pages 55–62.

[6] A.E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[7] A.E. Eiben, E. Marchiori, and V.A. Valko. Evolutionary Algorithms with on-the-fly Population Size Adjustment. In X. Yao et al., editor, *Parallel Problem Solving from Nature, PPSN VIII*, number 3242 in Lecture Notes in Computer Science, pages 41–50. Springer, Berlin, Heidelberg, New York, 2004.

[8] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[9] D.B. Fogel. Other selection methods. In Bäck et al. [3], chapter 27, pages 201–204.

[10] L. Gonzalez and J. Cannady. A self-adaptive negative selection approach for anomaly detection. In *2004 Congress on Evolutionary Computation (CEC'2004)*, pages 1561–1568. IEEE Press, Piscataway, NJ, 2004.

[11] P.J.B. Hancock. A comparison os selection mechanisms. In Bäck et al. [3], chapter 29, pages 212–227.

[12] N. Hansen, A. Gawelczyk, and A. Ostermeier. Sizing the population with respect to the local progress in (1, lambda)-evolution strategies. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 80–85. IEEE Press, 1995. (Authors in alphabetical order).

[13] G. Hardin. The tradegy of the commons. *Science*, 162:1243–1248, December 1968.

[14] M. Herdy. The number of offspring as a strategy parameters in hierarchically organized evolution strategies. *ACM SIGBIO Newsletter*, 13(2):2–9, 1993.

[15] S.W. Mahfoud. Boltzmann selection. In Bäck et al. [3], chapter 26, pages 195–200.

[16] E. Poupaert and Y. Deville. Acceptance driven local search and evolutionary algorithms. In L. Spector, E. Goodman, A. Wu, W.B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 1173–1180. Morgan Kaufmann, 2001.

[17] W.M. Spears. *Evolutionary Algorithms: the role of mutation and recombination*. Springer, Berlin, Heidelberg, New York, 2000.

[18] E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In E.K. Burke X. Yao, J.A. Lozano, J. Smith, J.J.M. Guervos, J.A. Bullinaria, J.E. Rowe, P. Tino, A. Kaban, and H.-P. Schwefel, editors, *PPSN*, number 3242 in Lecture Notes in Computer Science, pages 832–842. Springer, Berlin, Heidelberg, New York, 2004.

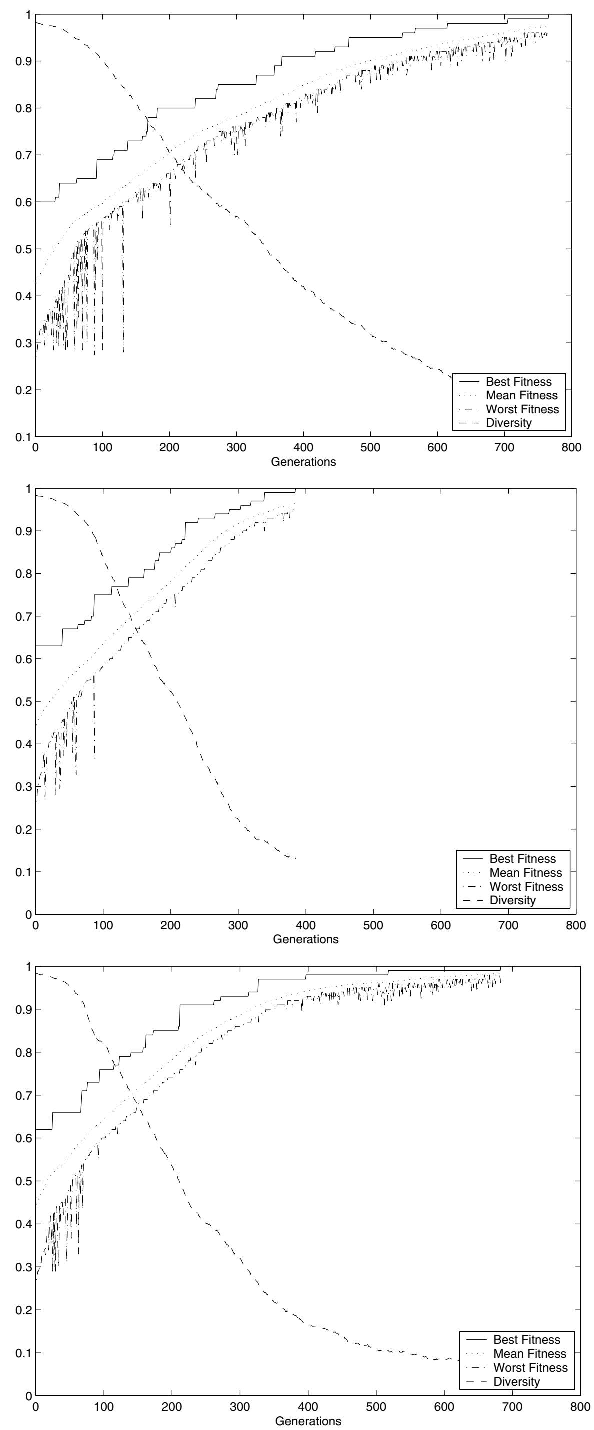

Fig. 2. A successful run with 10 peaks of SGA (top), GAHSAT (middle) and GASAT (bottom).