Introducing the Cumulation to the Population Based Incremental Learning and the Compact GA to Relax Genetic Drift

Keigo Tanaka
Graduate School of Science and Technology
Shinshu University
4-17-1 Wakasato
Nagano 380-8553, Japan
16w2044c@shinshu-u.ac.jp

Youhei Akimoto
Faculty of Engineering
Shinshu University
4-17-1 Wakasato
Nagano 380-8553, Japan
y_akimoto@shinshu-u.ac.jp

ABSTRACT

The population based incremental learning (PBIL) and the compact genetic algorithm (cGA) are both Bernoulli distribution based search algorithm for optimization of binary variables. The probability parameter that controls the probability of each bit being one is updated based on the ranking of the population from the current distribution. We often observe some parameters move randomly and tend to converge towards undesired values. Such a behavior is called genetic drift and it happens due to a large variance of the parameter update compared to a small expectation of the parameter update. To prevent genetic drift, the learning rate for the parameter update needs to be sufficiently small, but it results in a slow convergence. In this paper, we propose a mechanism to detect genetic drift and prevent parameters from being updated randomly. For this purpose, we introduce the so-called cumulation that is employed in the covariance matrix adaptation evolution strategy. Experimental results show that the cumulation allows PBIL and cGA to use a greater learning rate for the parameter update, leading to a speed up, especially on functions with high redundancy such as LeadingOnes function.

CCS CONCEPTS

•Mathematics of computing → Evolutionary algorithms; •Theory of computation → Evolutionary algorithms;

KEYWORDS

Population based incremental learning, compact genetic algorithm, binary optimization, genetic drift, cumulation

ACM Reference format:
Keigo Tanaka and Youhei Akimoto. 2017. Introducing the Cumulation to the Population Based Incremental Learning and the Compact GA to Relax Genetic Drift. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 2 pages.
DOI: http://dx.doi.org/10.1145/3067695.3075987

GECCO '17 Companion, Berlin, Germany
 978-1-4503-4939-0/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3067695.3075987

1 FORMULATION

Without loss of generality, we consider the minimization of $f : \{0,1\}^N \to \mathbb{R}$. We describe the population based incremental learning (PBIL), and derive the compact genetic algorithm (cGA) from the PBIL framework.

PBIL is a probability model based search algorithm. It produces multiple candidate solution from a multivariate Bernoulli distribution B_p with the parameter vector $p = (p_1, \ldots, p_N)$. The probability parameter is initialized as $p_i = 0.5$ for all $i = 1, \ldots, N$. Note that $p_i = 0.5$ implies no apriori preference between 0 and 1. PBIL repeats the following steps until a stopping condition is satisfied.
STEP 1. Sample λ candidate solutions $x_j \in \{0,1\}^N$ $(j = 1, \ldots, \lambda)$ from the Bernoulli distribution B_p
STEP 2. Evaluate the objective values $f(x_j)$
STEP 3. Update the probability parameter p as follows

$$p \leftarrow p + c_p \textstyle\sum_{j=1}^{\lambda} w_j (x_j - p) \ , \tag{1}$$

where c_p is the learning rate for p.
STEP 4. Restrict the probability parameter within $[\epsilon, 1-\epsilon]$ as

$$[p]_i \leftarrow \min(\max([p]_i, \epsilon), 1-\epsilon) \ , \tag{2}$$

where $\epsilon = 1/\lambda/N$ in this paper.

Compact Genetic Algorithm (cGA) [3] is a probability model based search algorithm for binary optimization. It samples two candidate solutions from B_p and update the probability parameters by adding $\pm c_p$ if one candidate solution is better than the other, and stall the update if they are tie. The cGA is fully recovered in the PBIL framework by setting $\lambda = 2$, $\tilde{w}_1 = 1/2$, $\tilde{w}_2 = -1/2$.

2 INTRODUCING THE CUMULATION IN PBIL

Issue of Genetic Drift. Both PBIL and cGA suffer from the genetic drift. It happens when the expectation of the parameter update $dp = \sum_{j=1}^{\lambda} w_j (x_j - p)$ is small relative to the standard deviation of the update. In this case the parameter is updated randomly without reflecting the right signal, i.e., $\mathbb{E}[dp]$.

It is known that a small learning rate, c_p, helps to prevent the genetic drift. The reason is explained as follows from the stochastic approximation point of view. Assume that c_p is so small that the parameter will not change significantly in T steps. Then, the parameter updates, dp, for T iterations are considered from the same distribution. Then, we have the expectation and the standard deviation of $[p^{(t+T)}]_i - [p^{(t)}]_i$ as $\mathbb{E}[[p^{(t+T)}]_i - [p^{(t)}]_i] = T c_p \mathbb{E}[[dp]_i]$

and $\mathrm{Std}[[p^{(t+T)}]_i - [p^{(t)}]_i] = \sqrt{T}c_p\mathrm{Std}[[dp]_i]$, respectively. Therefore, the expectation, i.e., the right signal, will be $\sqrt{T}$ times more emphasized than the standard deviation, i.e., the noise effect. Considering $T \propto 1/c_p$, one can conclude that the signal to noise ratio is proportional to $1/\sqrt{c_p}$.

However, the right c_p differs from problem to problem and from situation to situation even on the same problem since $\mathbb{E}[[dp]_i]$ and $\mathrm{Std}[[dp]_i]$ varies. For example, there will be almost no signal for the ith component for $i \approx N$ at the beginning of the search on LeadingOnes. On the other hand, we will receive stronger signal for all components on OneMax. In general, one can not know how small c_p should be, c_p needs to be set as small as possible, but it results in taking long iterations until the parameters converge.

PBIL with Cumulation. We introduce the evolution path $s \in \mathbb{R}^d$. It is initialized as $[s]_i = 0$ for $i = 1, \ldots, d$. Our proposed algorithm replaces **STEP3** with the following steps.

STEP3a. The evolution path s is updated as

$$s \leftarrow (1 - c_s)s + \sqrt{c_s(2 - c_s)\mu_v} \sum_{j=1}^{\lambda} v_j(x_j - p) \ , \qquad (3)$$

where c_s is the cumulation factor, v_j are the recombination weights for the evolution path computed in the same way as w_j with predefined weights $\bar{v}_j$ instead of $\bar{w}_j$, $\mu_v = 1/\sum_{j=1}^{\lambda} \bar{v}_j^2$.

STEP3b. The p_i is updated according to (1) if it satisfies $s_i^2 > \alpha\gamma_i^{(t+1)}p_i(1 - p_i)$, where γ_i is the correction factor updated as $\gamma_i \leftarrow (1 - c_s)^2\gamma_i + c_s(2 - c_s)$ with the initial value $r_i = 0$.

Recombination Weights. In the original paper of PBIL [1] the recombination weights are all positive and exponentially decreasing from the first $\bar{w}_1$ to the last $\bar{w}_\lambda$. On the other hand, cGA uses a positive $\bar{w}_1$ and a negative $\bar{w}_2$. To prevent undesired update, utilizing both positive and negative weights has a certain advantage. The following two schemes are considered.

$$\bar{w}_i, \bar{v}_i = \tfrac{1}{\mu} \ \ (\text{if } j \le \mu), \ \ 0 \ \ (\text{otherwise}) \qquad (4)$$

$$\bar{w}_i, \bar{v}_i = \tfrac{1}{2\mu} \ \ (\text{if } j \le \mu), -\tfrac{1}{2\mu} \ \ (\text{if } j \ge \lambda - \mu), 0 \ \ (\text{otherwise}) \qquad (5)$$

3 EXPERIMENTS

Table 1a and Table 1b show the average numbers of function evaluations over 50 trials until cGA and PBIL with and without cumulation find the optimum on 500 dimensional LeadingOnes. The maximum number of function evaluations is 5×10^6.

Both cGA and PBIL requires a smaller c_p to solve LeadingOnes efficiently. If one increases c_p, we often observe many components of the parameter drift towards 0, which is opposite to the optimal direction. On OneMax such a drift is less likely to happen, as we see in Figure 1. We also note that the negative weights (5) helps to speed up the optimization. In particular on OneMax, using the negative weights allows us to set a higher c_p and it results in a faster convergence.

Figure 1 compares the evolution of the probability parameters $[p]_i$ with and without cumulation. The same learning rate $c_p = 10/N$ is used for all settings. On LeadingOnes, we observe that most of the components of the probability parameter p move only toward 1 from the initial value of 0.5 when we introduce the cumulation ($\alpha = 11$), whereas we observe many components move towards 0 due to nearly random update of the parameters. Even though c_p is untouched, we observe the speed up. On the other hand, we observe a speed down on OneMax by introducing the cumulation. It is because the random parameter update is less likely to happen on OneMax and the proposed cumulation mechanism tends to stale the parameter update even in the right direction. Note that, however, one can set a greater c_p and a smaller α in the proposed algorithm, which will make the difference between PBIL with and without cumulation smaller on OneMax.

Table 1: Comparison of cGA, PBIL with (5) and PBIL with (4).

(a) without cumulation ($\alpha = 0$)

c_p	$1/N$	$5/N$	$10/N$	$50/N$	$100/N$
cGA	169	71.8	59.2	52.8	> 500
PBIL with (5)	146	45.6	68.9	111	127
PBIL with (4)	109	53.7	89.4	122	124

(b) with cumulation ($\alpha = 11$)

c_p	$1/N$	$5/N$	$10/N$	$50/N$	$100/N$
cGA	126	34.4	26.0	> 500	> 500
PBIL with (5)	163	38.3	21.3	6.51	5.88
PBIL with (4)	126	30.9	17.6	5.79	5.15

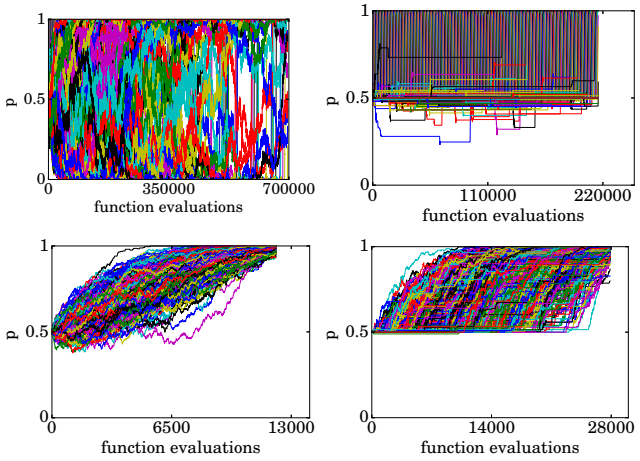

Figure 1: PBIL without cumulation ($\alpha = 0$, left) and PBIL with cumulation ($\alpha = 11$, right) with (5), $c_p = 10/N$, $c_p/c_s = 5$ on 500 dimensional LeadingOnes: $f(x) = N - \sum_{j=1}^{N}\prod_{i=1}^{j}[x]_i$ (upper) and OneMax: $f(x) = N - \sum_{i=1}^{N}[x]_i$ (lower).

REFERENCES

[1] Shumeet Baluja and Rich Caruana. 1995. Removing the genetics from the standard genetic algorithm. In *Proc. of the 12th Intern. Conf. on Machine Learning*. 38–46.

[2] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation* 9, 2 (2001), 159–195.

[3] Georges R Harik, Fernando G Lobo, and David E Goldberg. 1999. The compact genetic algorithm. *IEEE transactions on evolutionary computation* 3, 4 (1999), 287–297.

[4] Dirk Sudholt and Carsten Witt. 2016. Update strength in EDAs and ACO: How to avoid genetic drift. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM, 61–68.