Design of an Adaptive Push-Repel Operator for Enhancing Convergence in Genetic Algorithms

Yashesh Dhebar
Department of Mechanical Engineering
Michigan State University
East Lansing, USA
dhebarya@egr.msu.edu

Kalyanmoy Deb
Department of Electrical and Computer Engineering
Michigan State University
East Lansing, USA
kdeb@egr.msu.edu

COIN Report Number 2018007
www.coin-laboratory.com

***Abstract*—Genetic Algorithms (GAs) are demonstrated to be successful in solving problems pertaining to the field of engineering, physics, medicine, finance and many more. The efficacy of GAs lies in its efficiency at exploring complex design-space with black-box constraints and reach the optimal regions defined by functions of unknown fitness landscapes (or in other words, black-box optimization functions). Depending on the nature of the problem, the design-space can have continuous, discrete or mixed (continuous and/or discrete) set of design-variables. The exploration in this design-space is conducted through a population of individuals and is primarily driven by three operations – *selection, recombination (or crossover)* and *mutation*. The *exploitation* aspect of a GA search is obtained by its selection operation, while crossover and mutation operations deal with the *exploration* aspect for generating new solutions in the search space. In this study, an attempt has been made to balance the two aspects by designing a generic *push* operator which introduces an extra level of exploitation in the algorithm by biasing the creation of solutions near the best-so-far solution. In addition to standard search operators, an additional diversity maintaining *repel* operator is introduced to balance the exploitation-exploration issue. Simulations are performed to understand the effect of an adaptive push-repel GA on different fitness landscapes for both unconstrained and constrained optimization problems. The results are promising and encourage their extensions to other evolutionary algorithms.**

***Index Terms*—Optimization, genetic algorithm, push operator, non-uniform mapping.**

I. Introduction

The success of evolutionary algorithms or any other population based approaches lies in their ability to smartly navigate through complex fitness landscapes in a quest to reach and determine an optimal solution (for unimodal single objective optimization problems) or a set of optimal-solutions(for multimodal or multiobjective optimization problems). In case of genetic algorithms, the search is mainly driven by three operators, viz *selection, crossover* and *mutation*. Selection and crossover operations are influenced by the spread of population in the design space. The selection operation implements a strategy to select a set of good points from the population pool. In GA literature, some of the popular approaches for selection are *proportionate selection* [1], *ranking selection* [2], *tournament selection* [3], etc. The selected set of population individuals undergo cross-over (i.e. recombination), to generate a child population. One of the widely used cross-over approach for handling real-variables is *simulated-binary-crossover* (or SBX in short) [4]. The SBX operator uses a polynomial probability distribution function, which is spiked at the location of two participating parent solutions. This probability density function dictates the probability of the location of two child-solutions. There is a high probability for the separation between two generated child-solutions to be similar with the distance between corresponding parent solutions. Hence, the diversity of the population is sustained after the cross-over operation. The population is then subjected to subtle perturbations through mutation-operation. In this paper, we have used polynomial mutation [5]. The crossover and mutation together fundamentally focuses at the *exploration* task by creating new set of points in the search space, while the selection-operation introduces *exploitation* of current best solutions by driving the algorithm towards *convergence*. Though extensive research has been made at designing new selection, crossover or mutation operators ([6]–[9]) to make a balance between the two aspects, a very little effort has been made in exploiting the population information to bias the search towards potentially better regions in the search space.

In the remainder of this paper, we provide a brief literature review of exploitation-specific operators in Section II. In Section III, we present our original non-uniform mapping concept which has led us to the generic push-repel operator, a detail description of which is provided in Section IV. Section V presents the results of our proposed procedure. Conclusions of this study are made in Section VI.

II. Existing Studies on Non-Uniform Mapping in GAs

Work done in [10] (ARGOT) introduces the bias in search-space with an adaptive mapping approach for binary to decimal conversion. It uses parameter-statistics such as variance, position, etc and several environmentally triggered operators to alter intermediate mappings. Dynamic parameter encoding (DPE) [11] implements a hierarchical strategy to conduct the *global* and *local* search. In the first phase, global search is realized by making updates on the most-significant bits. Local search is then executed by appending extra bits for higher precision and dropping the most-significant bits. Delta Encoding approach developed in [12] utilizes the information about the location of the partial optimal solution attained so far to construct a population for next iteration. A new hyper-cube surrounding the best-so-far solution is formed by generating points through Δ perturbations on the most recent partial solution.

In this paper, we extend the idea of *non-uniform-mapping* [13], [14] and propose a generic *push* operator which is used to *push* the population towards regions with better fitness values. The push-operation implicitly mimics gradient-based-search[1] by providing a directional aspect in generating set of new points. The search-bias resulting from the push-operation aids in attaining faster convergence. In next section, a brief overview on *non-uniform-mapping* is presented. In remaining of the paper, the methodology to design a *push-repel* operator has been elaborated. The idea is then tested on set of unimodal and multi-modal single objective optimization problems on domains involving boxed or generic constraints.

III. Non-Uniform Mapping in Binary Coded GAs

Non-uniform mapping was first developed for binary coded GA in [13]. The idea was to device a non-linear *binary-to-real* decoding scheme. This was realized through a polynomial distribution function as shown in Figure 1.

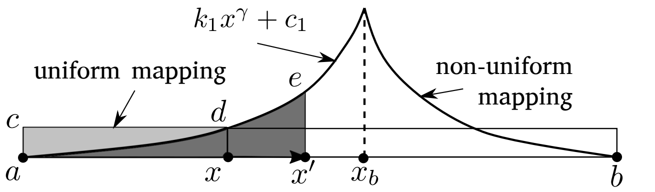


Figure 1: Non-uniform mapping: x is obtained using a regular binary-to-decimal decoding (which translates to uniform-mapping between bounds a and b). x' is obtained by implementing a non-uniform mapping to convert binary-string to a real-number.

The mapped value x' is obtained by equating the area under the *uniform-mapping-curve* (Area$(acdx)$) with the area under a *non-uniform mapping curve* (Area$(aex'a)$). The polynomial distribution function of degree γ peaks at the best-so-far point x_b, and x_b is constrained to map to itself, i.e. $x'_b = x_b$. The mapping thus favors creation of more decoded points near x_b as shown in Figure 2.

IV. Design of a Generic Push Operator

The idea of non-uniform mapping, when applied to a real-coded genetic algorithm, translates to a push-operation. Upon invoking the push-operation on the population, each individual gets *pushed* towards the best-so-far point $\mathbf{x_b}$ as per the following equation:

$$x'^{(i)} = \begin{cases} x_L^{(i)} + [(x_b^{(i)} - x_L^{(i)})^\gamma (x^{(i)} - x_L^{(i)})]^{\frac{1}{1+\gamma}} & x^{(i)} \le x_b^{(i)} \\ x_U^{(i)} - [(x_U^{(i)} - x^{(i)})(x_U^{(i)} - x_b^{(i)})^\gamma]^{\frac{1}{1+\gamma}} & x^{(i)} > x_b^{(i)} \end{cases} \tag{1}$$

[1]In gradient-based approaches, the direction of search is determined by the direction of gradient at a point under consideration.

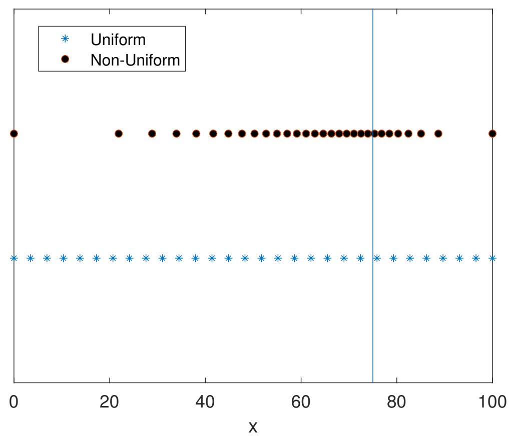


Figure 2: The best-so-far is located at $x = 75$. The equi-spaced points (marked by $*$) are created through a regular binary-to-real decoding, while the non-uniformly spaced point (marked by a solid circle) are generated using decoding resulting from non-uniform mapping.

where i indicates the i^{th} component of a variable vector and subscripts L and U stands for lower and upper bound respectively. The location of an individual, $\mathbf{x}$, thus gets *modified* to $\mathbf{x}'$. This idea was tested on some unconstrained single objective optimization problems in [14], [15] and [16]. This idea of push-operator is further explored in the current study with an intention to attain faster convergence under varied fitness-landscapes for both, unconstrained and constrained optimization problems. In all the experiments with a real-coded GA (RGA) and all push operator based GAs developed in this study, SBX crossover, polynomial mutation and binary tournament selection operators are used. Parameters setting common to all unconstrained optimization problems is as follows: SBX distribution parameters $\eta_c = 2$, polynomial mutation distribution parameter $\eta_m = 15$, crossover probability $p_c = 0.9$, mutation probability $p_m = 1/n_{var}$, population size (N_p)= 100, number of variables (n_{var}) = 20, maximum generations (g_{max}) = 1000. 50 runs are executed for each problem. A run is reported as a success if accuracy within 0.01 error in objective space is attained. For a successful runs (S), statistics for number of function evaluations (FE) are reported. For failed runs (F), statistics for the terminating function-values (FV) are reported.

In the following subsections, a systematic development of the overall push operator based genetic algorithm is presented.

A. Effect of Survival Selection Strategy

Since the push-operation inherently captures greediness by biasing the search towards $\mathbf{x_b}$, any extra greediness (or exploitation) might be detrimental, in general. But a controlled operation of the push-operator may be beneficial, which is what we develop as a push-enabled/disabled GA. Two main survival strategies considered for experimentation are (μ, λ) and $(\mu + \lambda)$-EAs. Simulations are performed on $20D$ Ackley's function with a recommended rate of increase in γ of 0.033 per generation. Results for the same are tabulated in Table I. Results clearly indicate that for a push-enabled GA, (μ, λ) is better than $(\mu+\lambda)$ survival strategy. However, $(\mu+\lambda)$ becomes conducive to attain better convergence in regular (i.e. push-disabled) GA. In all future experiments, (μ, λ) approach is used for push-enabled GA, and, $(\mu + \lambda)$ survival strategy is adopted for runs executed with regular (push-disabled) GA.

Table I: Results on Ackley's Function to understand the effect of survival-strategies and push-enabled GA. 50S means all 50 runs are successful and 50F indicates none of the 50 runs is successful. In successful runs, function evaluations are mentioned and for unsuccessful runs the achieved objective function values are mentioned.

Push	**Survival**	**S/F**	**min**	**avg**	**med**	**max**
RGA (no push)	$(\mu+\lambda)$	**50S**	**12600**	**15752**	**15100**	**23700**
	(μ,λ)	50F	0.34	0.54	0.55	0.70
Push Enabled	$(\mu+\lambda)$	50S	9100	10888	10750	13600
	(μ,λ)	**50S**	**6700**	**7686**	**7700**	**9000**

B. Effect of the Location of Optimum

Push operation creates a bias in the search space by pushing population towards the best-so-far point $\mathbf{x_b}$. This aids in conducting a focused search near $\mathbf{x_b}$ but at the cost of depleted diversity. It is advantageous when $\mathbf{x_b}$ is surrounded by other population members and is approximately centered relative to the population pool as shown in Figure 3a. Here, the push-operation tends to create points near to the global optimum $\mathbf{x_{opt}}$. Another situation which favors faster convergence to the optimum is when the push-operation results into creation of points inside the region bounded by a hyper-sphere of radius $\|\mathbf{x_{opt}} - \mathbf{x_b}\|$, centered at $\mathbf{x_{opt}}$ (where $\mathbf{x_{opt}}$ is the true optimum) as shown in Figure 3b. However, if the population cloud and true optimum are separated by the tangent-hyperplane (as shown in Figure 3c), the push-operation results into depletion in convergence rate. This is because, in addition to the deteriorated population diversity, the push-operation fails to create points closer to $\mathbf{x_{opt}}$ than $\mathbf{x_b}$. This situation restricts the search capability and also results into premature convergence of the algorithm.

From the above discussion, it can be stated that the ability of push-operation to create points closer to $\mathbf{x_{opt}}$ than $\mathbf{x_b}$ facilitates a faster convergence. To validate this hypothesis, experiments are performed on 20D Ackley's Function with optimum at the origin and the domain is adjusted such that the optimum can lie either

- at the *center* of the domain: *CENTERED*, or
- near to the center of the domain: *APPROX. CENTERED*, or
- on the corner of the domain: *ONESIDED*, or
- near the corner of the domain *APPROX. ONESIDED*.

The population distribution similar to the one shown in Figures 3a and 3b is often encountered when the optimum is *centered* (or approximately centered) while the situation similar to the one shown in Figure 3c mostly pertains when the optimum is located at the domain-corner. Results of these experiments are shown in Table II.

Table II: Results on Ackley's Function to understand the effect of the location of the optimum and population distribution relative to the best-so-far point. The location of optimum is at the origin. Number in brackets in the first column indicates lower and upper bounds. RGA results are without any push.

Case	**Push**	**S/F**	**min**	**med**	**max**
CENTERED $[-5, 5]$	RGA	50S	12600	15100	23700
	Enabled	**50S**	**6700**	**7700**	**9000**
APPROX. CENTERED $[-4, 6]$	RGA	50S	12000	15400	20800
	Enabled	**50S**	**6800**	**7750**	**9000**
ONESIDED $[0, 10]$	RGA	**50S**	**9500**	**10100**	**11000**
	Enabled	50F	3.13	3.49	3.73
APPROX. ONESIDED $[-0.3, 9.7]$	RGA	**50S**	**12400**	**14100**	**19500**
	Enabled	28S	14400	18750	30200
		22F	1.42	2.09	2.45

C. Simulated Centrality

It is observed in the previous section that the push-enabled GA works better when the optimum is situated at the center of the population (*centeredness*), i.e. the situation where the best-so-far point is approximately at the center of the population cloud, or, most of the population members lie on the same side of the tangent-plane as the true-optimum $\mathbf{x_{opt}}$ (Figure 3b). In order to simulate centeredness, a *mirroring technique* is implemented, wherein the pushed point $\mathbf{x'}$ gets reflected/mirrored with respect to $\mathbf{x_b}$ and assumes a new location $\mathbf{x'_m}$ as shown in Figure 4a. If $\mathbf{x'_m}$ falls outside the domain boundary, it is repaired by projecting the violated component of $\mathbf{x'_m}$ to its corresponding nearest domain wall. This idea is illustrated in Figure 4b. In the proposed algorithm, randomly 50% of population individuals get mirrored.

D. Escaping a Local Attractor

Push-operation is susceptible to premature convergence near a local optimum due to depletion in population diversity. A result to validate this argument can be seen in 2^{nd} row of Table III, where the push-operation fails the algorithm to converge in all 50 runs on multi-modal Rastrigin's function. This issue of premature convergence to a local optimum can be addressed by repelling away a set of population individuals from $\mathbf{x_b}$. The location of the repelled point $\mathbf{x'_r}$ is determined by following equations:

$$x_r^{(i)} = \begin{cases} x_b^{(i)} + [(x_U^{(i)} - x_b^{(i)})^{\gamma}(x^{(i)} - x_b^{(i)})]^{\frac{1}{1+\gamma}} & x^{(i)} \geq x_b^{(i)} \\ x_b^{(i)} - [(x_b^{(i)} - x^{(i)})(x_b^{(i)} - x_L^{(i)})^{\gamma}]^{\frac{1}{1+\gamma}} & x^{(i)} < x_b^{(i)} \end{cases} \tag{2}$$

By relating Eq. 2 with Eq. 1, it can be stated that repelling an individual away from $\mathbf{x_b}$ is equivalent to pushing an individual towards the domain walls.

The repel operation is invoked when the algorithm stagnates and the change in function value of $\mathbf{x_b}$ is less than a pre-specified threshold, which in our case is set to 0.1%, i.e. when

$$\frac{f(x_b^{(\text{prev})}) - f(x_b^{(\text{curr})})}{(f(x_b^{(\text{prev})})} \leq 0.001. \tag{3}$$

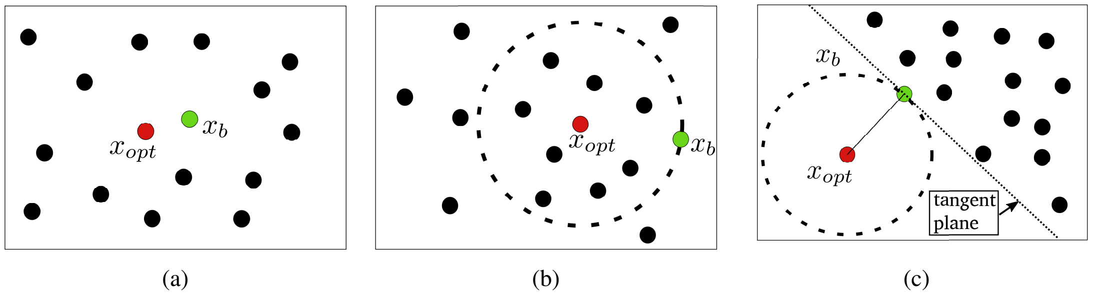


Figure 3: Locations of the true optimum $\mathbf{x_{opt}}$ and the best-so-far point $\mathbf{x_b}$ with respect to the population pool. In 3a, true optimal point $\mathbf{x_{opt}}$ and the best-so-far point $\mathbf{x_b}$ are centered relative to the population. In 3b, the true optimum $\mathbf{x_{opt}}$ is centered but $\mathbf{x_b}$ is not. In 3c, neither of $\mathbf{x_{opt}}$ or $\mathbf{x_b}$ are centered. The tangent plane separates the population pool from $\mathbf{x_{opt}}$.

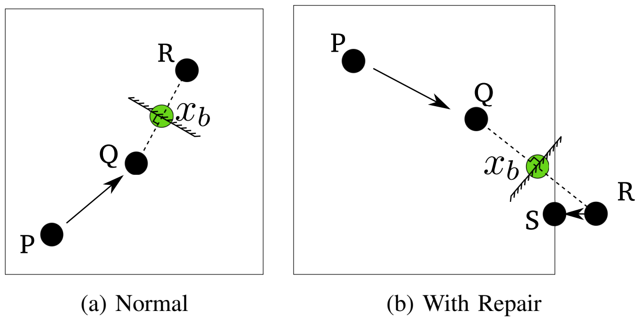


Figure 4: Mirroring to simulate centrality. Point P is pushed towards $\mathbf{x_b}$ using Eq. 1 and assumes a new location Q. Point Q is then mirrored with respect to $\mathbf{x_b}$ to obtain R as shown in (a). If R violates domain bounds, it is repaired and set to location S as shown in (b) to make the solution feasible.

```
Data: Population $Q_\lambda$, best-so-far point $\mathbf{x_b}$, $\lambda$ is the
      population size.
PARTITION: $Q_\lambda \rightarrow Q_{\frac{\lambda}{2},near} \cup Q_{\frac{\lambda}{2},far}$
for $q \in Q_{\frac{\lambda}{2},near}$ do
    $q \rightarrow q'$... push using Eq. 1
    if rand() < 0.5 then
        mirror($q'$).... according to Figure 4
    end
end
for $q \in Q_{\frac{\lambda}{2},far}$ do
    $q \rightarrow q_r$... repel using Eq. 2
    if rand() < 0.5 then
        mirror($q_r$).... according to Figure 4
    end
end
```

Algorithm 1: Pseudo code for push-repel operation.

When the above criteria is met, distant 50% population-individuals are repelled away from $\mathbf{x_b}$. The closer 50% population members however undergo the usual push. It is to note here that the mirroring is enabled for both (push and repel) operations with 50% probability. The γ for repel-operation is upper bounded by a threshold value of $\tau_r = 0.1$. A pseudo-code of the push-repel algorithm is provided below (Algo. 1). The implementation of the push-repel operation aids in attaining convergence on multi-modal landscapes like Rastrigin's function as shown in Table III.

E. APR-GA: A Population-based Adaptive Push-Repel GA

The above idea seems to work well for unconstrained problems. However, the results are not so fruitful for simulations performed on constrained optimization problems. An example of it is shown in Table IV for the $G1$-function [17]. One of the reasons a regular RGA performs better is its broad *exploratory* nature. The push-operation, being more greedy in its core (due to biasing created towards $\mathbf{x_b}$), it emphasizes *exploitation* more than exploration. For initial generations, it is essential to maintain diversity in population for better exploration of search space; and eventually, it is favorable to do a localized focused search. One way to achieve this is by combining a regular RGA (without push) and a push-enabled RGA to harness advantages of both and attain better performance. In the proposed hybrid algorithm, shown in Figure 5, diversity

Table III: Results on Rastrigin's function to understand the effect of the push operator on a complex multi-modal fitness landscape.

Case	Push Type	S/F	min	med	max
Centered	No Push (RGA)	50S	21000	29400	42700
	Only Push	50F	8.95	34.82	59.70
	Push + Repel	**50S**	**20000**	**25050**	**37400**

Table IV: Results on G1 Function to understand the effect of the push operator on a constraint optimization problem. *APR-GA* indicates our final adaptive push-repel genetic algorithm proposed in Figure 5.

Function	Push	S/F	min	med	max
G1 (Constrained)	RGA	47S	8710	10140	19110
		3F	2.00	2.55	2.55
	Push + Repel	11S	3380	31850	65130
		39F	1.18	3.00	8.00
	APR-GA	**47S**	**4810**	**6500**	**17030**
		3F	2.00	2.55	2.55

of the population in terms of normalized-standard-deviation[2] along each dimension is monitored. The maximum standard deviation $\sigma_{max} = \max(\sigma_1, \sigma_2, \ldots, \sigma_d)$ (where $d = n_{var}$ is the dimension of the search space) is then used as a *flag* to trigger the push-operation. Push-operation is invoked when $\sigma_{max} < 0.1$, and, the algorithm permanently switches from a regular push-disabled GA (with $(\mu + \lambda)$ survival strategy) to a push-enabled GA. Also, within the push-operation, only closest 50% of the population members from $\mathbf{x_b}$ undergo push (or push + random-mirroring); while the rest either retain their location or are repelled according to the condition given in Eq. 3. After successful execution of the push-operation, *pushed* population members (and repelled population members) are subjected to (μ, λ) survival approach and *un-pushed* members are combined with the parent population to undergo $(\mu + \lambda)$ survival selection. The flowchart of the overall algorithm is schematized in Figure 5.

The above-proposed adaptive push-repel GA (APR-GA) uses a higher exploitation of current-best solutions to expect a faster convergence, but to balance the added exploitation, an increased but carefully regulated exploration is introduced by means of a repelling strategy. As the simulation results will demonstrate in the next section, this increased exploitation-exploration strategy constitutes a better overall optimization procedure than that without the push-repel strategy.

V. Results

Experiments are conducted on five unconstrained and eight constrained optimization problems.

A. Unconstrained Problems

For unconstrained problems, two scenarios are considered: (i) Optimum at the center of the domain (C), and (ii) Optimum at the corner of the domain (O). The objective function chosen for benchmarking the algorithms are mentioned below:

- Sphere: $\sum_{i=1}^{D} x_i^2$
- Ellipsoidal: $\sum_{i=1}^{D} i x_i^2$
- Ackley: $-20 \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{D} x_i^2}\right)$
$$- \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + 20 + e,$$
- Schwefel: $\sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_i\right)^2$
- Rastrigin: $10D + \sum_{i=1}^{D}[x_i^2 - 10cos(2\pi x_i)]$

Variable bounds are set $[-5, 5]$ for problems having optimum at the center and $[0, 10]$ for problems having optimum at the corner. $D = 20$ dimensional problems are chosen. 50 runs are conducted, and a run is considered a success (S) if a solution with an objective value within 0.01 from the true optimal objective value is achieved. For successful runs, statistics in terms of *minimum, median and maximum* number of function evaluations (FEs) are reported, while for failed runs, *minimum, median and maximum* function values (FV) are reported. The results on unconstrained problems are shown in Table V.

[2]where normalization is done by dividing the *standard-deviation* with the *range* of the variable (i.e. $x_U^{(i)} - x_L^{(i)}$)

Table V: Results on unconstrained problems for finding one global minimum. 'C' indicates the optimum is located at the center of the domain, while 'O' indicates that the optimum is located at the corner of the domain.

Function	Push	S/F	min	med	max
Sphere (C)	RGA	50S	7200	8300	9300
	APR-GA	**50S**	**5600**	**6100**	**6800**
Sphere (O)	RGA	50S	6700	7300	8500
	APR-GA	**50S**	**2200**	**2800**	**3300**
Ellipsoidal (C)	RGA	50S	9400	11600	13900
	APR-GA	**50S**	**7300**	**8150**	**9300**
Ellipsoidal (O)	RGA	50S	8100	9050	9600
	APR-GA	**50S**	**2600**	**3500**	**4600**
Ackley (C)	RGA	50S	12600	15100	23700
	APR-GA	**50S**	**8900**	**10300**	**13500**
Ackley (O)	RGA	50S	9500	10100	11000
	APR-GA	**50S**	**2300**	**2800**	**3300**
Rastrigin (C)	RGA	50S	**21000**	29400	42700
	APR-GA	**50S**	21100	**26000**	**35800**
Rastrigin (O)	RGA	50S	17600	21350	32400
	APR-GA	**50S**	**5000**	**11300**	**15000**
Schwefel (C)	RGA	50F	1.70e-02	5.59e-02	1.04e-01
	APR-GA	**50S**	**21500**	**46800**	**78700**
Schwefel (O)	RGA	50S	7700	8500	9100
	APR-GA	**50S**	**2600**	**3400**	**4300**

It can be clearly concluded from the results that the suggested approach for pushing the population aids in attaining faster convergence for most of the problems. Considerable reduction in the number of function evaluations is observed when the optimum is lying at the corner of the domain. This is mostly due to the implementation of a *repair* operation on individuals violating domain bounds (shown in Figure 4b). To understand the dynamics of the algorithm, the value of *push-flag* is plotted across generations for centered ellipsoidal and centered Rastrigin's function as shown in Figure 6. A push-flag value of 0 indicates that the population is evolved using regular GA operators, 1 stands for the case where Q_{near} is pushed while Q_{far} is untouched and push-flag value of -1 indicates the situation where the *repel* operation is invoked on Q_{far}.

It can be seen that multi-modal landscape of Rastrigin function triggers multiple *only-push*-to-*push-repel* transitions before attaining convergence, while, for a unimodal landscape such as that of an ellipsoidal function, the algorithm is sustained at *push-repel* phase, thereby suggesting that the difference in the function values is always retained below 0.001% and the entire population is distributed around an optimal region. For further analysis, a trend of fitness values of $\mathbf{x_b}$ (shown in Figure 7) is investigated. Though the overall performance of APR-GA is better (as can be seen from results tabulated in Table V), push-enabled and push-disabled GA gives almost similar trend for ellipsoidal function. Since the push-operation is designed as a *population based local-optimizer* (Sec. IV-E), another study is conducted to check the *local-optimizer* aspect of the algorithm. The termination criteria is set to 1,000 generations and the statistics in terms of *minimum*, *median* and *maximum* function values (FV) at the end of 1,000-th generation for 50 runs are reported in Table VI.

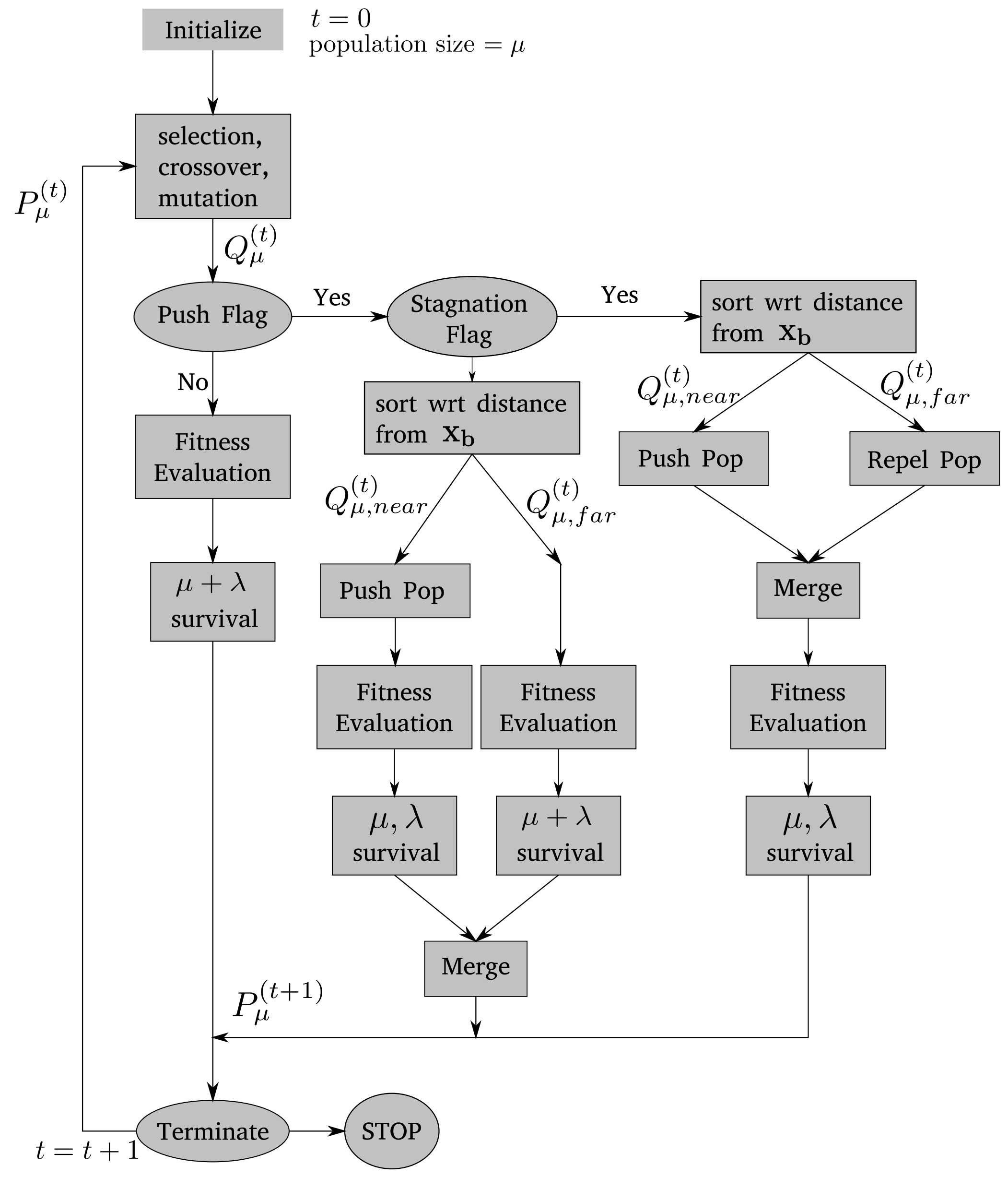


Figure 5: Flowchart of the overall adaptive push-repel operator based evolutionary algorithm (APR-GA).

Figure 8 shows the fitness value of $\mathbf{x_b}$ across 1000 generations.

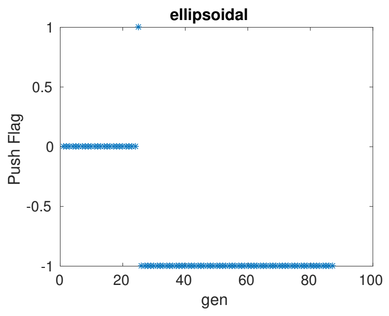
Table VI: Results on unconstrained problems for finding one global minimum with high accuracy.

Function	Push	S/F	min	med	max
Sphere (C)	RGA	50S	2.73e-10	4.80e-08	1.08e-06
	APR-GA	**50S**	**5.85e-13**	**1.25e-12**	**1.92e-12**
Ellipsoidal (C)	RGA	50S	1.45e-08	3.41e-07	9.98e-06
	APR-GA	**50S**	**6.92e-12**	**1.29e-11**	**2.03e-11**
Ackley (C)	RGA	50S	6.63e-05	3.50e-04	1.17e-03
	APR-GA	**50S**	**7.01e-07**	**9.91e-07**	**1.22e-06**
Schwefel (C)	RGA	50S	1.70e-02	5.59e-02	1.04e-01
	APR-GA	**50S**	**8.10e-06**	**1.69e-04**	**2.26e-03**
Rastrigin (C)	RGA	50S	2.61e-07	2.00e-05	2.80e-04
	APR-GA	**50S**	**1.50e-10**	**2.52e-10**	**3.84e-10**

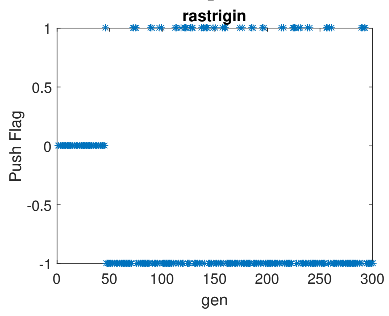
It is clearly evident from the plots of Figure 8 that the push-enabled GA (APR-GA) outperforms a regular RGA in terms of both: (i) attaining faster convergence and (ii) attaining a high accuracy. Except for the Schwefel's function, the trend of fitness value of $\mathbf{x_b}$ eventually assumes an asymptotic behavior for both RGA and APR-GA.

B. Constrained Problems

Simulations with APR-GA and RGA (without push) are performed for eight constrained optimization problems, viz. *crescent, G1, G4, G5, G6, G8, G12* and *G12*. Definitions for *G-functions* can be obtained from [17]. The crescent problem has be adopted from the *test-problem-1* in [18]. Deb's parameter-less constraint handling approach [18] is used to handle constraints. A run is considered successful when the accuracy within 1% of the optimal-fitness value is achieved

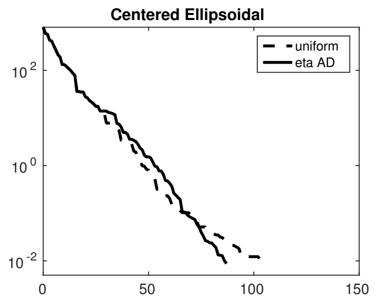


(a) Ellipsoidal

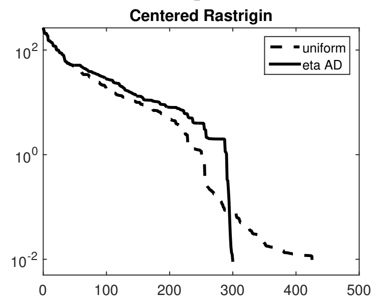


(b) Rastrigin

Figure 6: Push Flag dynamics across generations for optimum at the center of the domain. Figure 6a shows dynamics for unimodal ellipsoidal function and Figure 6b shows dynamics for multimodal Rastrigin's function.



(a) Ellipsoidal



(b) Rastrigin

Figure 7: Fitness value of best-so-far point $\mathbf{x_b}$ across generations for centered ellipsoidal (Figure 7a) and centered rastrigin (Figure 7b).

in the objective space[3]. The information about the dimension of the problem, population size and maximum number of generations is provided in Table VII. Rest of the parameters are kept fixed as before. Results for constrained-optimization problems are tabulated in Table VIII. APR-GA performs better than RGA in most problems.

Table VII: Parameter setting for constrained optimization problems.

Function	Dimension	Population Size	max gens
Crescent	2	50	50
G1	13	130	500
G4	5	50	5000
G5	4	100	1000
G6	2	50	1000
G8	2	20	1000
G9	7	70	1000
G12	3	30	1000

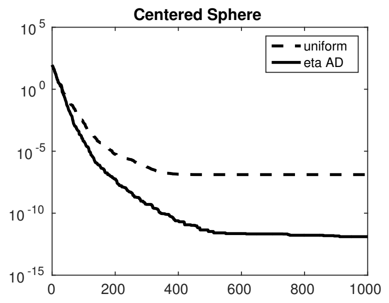
VI. Conclusions

In this paper, a systematic design of a push-operator based GA has been proposed. The overall algorithm has been structured in a hybrid form to encompass a genetic algorithm with a push-enabled operator (APR-GA) and harness their respective properties to attain a faster convergence. Results on unconstrained problems have shown that the proposed APR-GA has clearly dominated the standard RGA in terms of attaining a faster convergence and achieving a high accuracy on different fitness landscapes. The algorithm has also performed substantially well on constrained optimization problems. It is to note that throughout the development process of the algorithm, the central idea has been focused at broadening the exploration of search operators to counteract the large exploitation introduced by the push operator. This is achieved by creating a *careful*

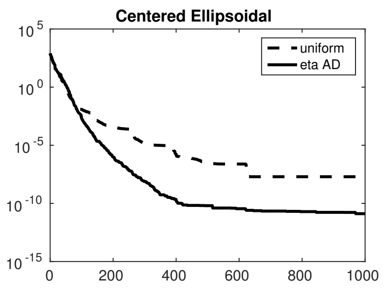
[3]for problems with optimal fitness value of 0, desired accuracy level is set to 0.01.

Table VIII: Results on constrained optimization problems.

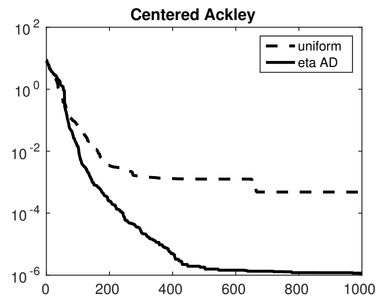
Function	Push	S/F	min	med	max
Crescent	RGA	29S	250	1100	2500
		21F	0.14	15.21	81.02
	APR-GA	**50S**	**250**	**750**	**1550**
G1	RGA	47S	8710	10140	19110
		3F	2.00	2.55	2.55
	APR-GA	**47S**	**4810**	**6500**	**17030**
		3F	**2.00**	**2.55**	**2.55**
G4	RGA	50S	350	1125	6800
	APR-GA	**50S**	**350**	**775**	**1400**
G5	RGA	2S	9700	10750	11800
		48F	71.44	1077.14	1975.21
	APR-GA	**16S**	**26000**	**44450**	**73000**
		34F	**62.92**	**468.57**	**2905.22**
G6	RGA	**24S**	**700**	**18675**	**48350**
		26F	70.04	90.22	156.29
	APR-GA	**50S**	**1100**	**4100**	**6900**
G8	RGA	**50S**	**40**	**80**	**120**
	APR-GA	**50S**	**40**	**80**	**120**
G9	RGA	50S	1960	2695	35350
	APR-GA	**50S**	**1820**	**2415**	**4480**
G12	RGA	**50S**	**120**	**405**	**840**
	APR-GA	50S	120	435	930



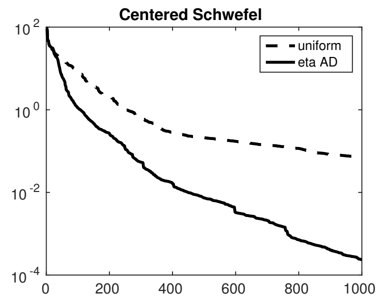
(a) Sphere



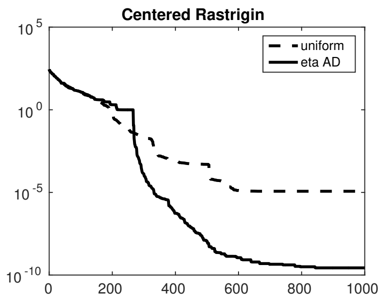
(b) Ellipsoidal



(c) Ackley



(d) Schwefel



(e) Rastrigin

Figure 8: Fitness values of best-so-far point across 1,000 generations for different unconstrained fitness landscapes. The optimum is at the center of the domain.

bias in the search space with an intention of moving population members closer to the true optimum $\mathbf{x_{opt}}$ than the best-so-far point $\mathbf{x_b}$. The discussion and results highlight a crucial important task in population-based algorithm design – while a large selection pressure is possible to implement for a better convergence and for a better utilization of derived information about the current-best solutions, this must come from a careful enhancement of exploration of the search operators. Such understanding allows EC researchers to push the boundary to achieve more efficient algorithms and encourages further implementation of similar concepts in other population based algorithms to improve their convergence properties.

REFERENCES

[1] J. H. Holland, "Adaptation in natural and artificial systems. an introductory analysis with applications to biology, control and artificial intelligence," *Ann Arbor: University of Michigan Press, 1975*, 1975.

[2] J. E. Baker, "Adaptive selection methods for genetic algorithms," in *Proceedings of an International Conference on Genetic Algorithms and their applications*, pp. 101–111, Hillsdale, New Jersey, 1985.

[3] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of genetic algorithms*, vol. 1, pp. 69–93, Elsevier, 1991.

[4] K. Deb, "Simulated binary crossover for continuous search space," *Complex systems*, vol. 9, pp. 115–148, 1995.

[5] K. Deb and M. Goyal, "A combined genetic adaptive search (geneas) for engineering design," *Computer Science and informatics*, vol. 26, pp. 30–45, 1996.

[6] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 4, pp. 656–667, 1994.

[7] H.-M. Voigt, H. Mühlenbein, and D. Cvetkovic, "Fuzzy recombination for the breeder genetic algorithm," in *Proc. Sixth Int. Conf. on Genetic Algorithms*, Citeseer, 1995.

[8] O. Hrstka and A. Kučerová, "Improvements of real coded genetic algorithms based on differential operators preventing premature convergence," *Advances in Engineering Software*, vol. 35, no. 3-4, pp. 237–246, 2004.

[9] N. L. Law and K. Y. Szeto, "Adaptive genetic algorithm with mutation and crossover matrices," in *20th IJCAI International Joint Conference on Artificial Intelligence*, p. 2330, 2007.

[10] C. G. Shaefer, "Argot strategy: adaptive respresentation genetic optimizer technique," in *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*, Hillsdale, NJ: L. Erlhaum Associates, 1987., 1987.

[11] N. N. Schraudolph and R. K. Belew, "Dynamic parameter encoding for genetic algorithms," *Machine learning*, vol. 9, no. 1, pp. 9–21, 1992.

[12] D. Whitley, K. Mathias, and P. Fitzhorn, "Delta coding: An iterative search strategy for genetic algorithms," in *ICGA*, vol. 91, p. 77e84, 1991.

[13] K. Deb, Y. D. Dhebar, and N. Pavan, "Non-uniform mapping in binary-coded genetic algorithms," in *Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012)*, pp. 133–144, Springer, 2013.

[14] D. Yashesh, K. Deb, and S. Bandaru, "Non-uniform mapping in real-coded genetic algorithms," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2237–2244, IEEE, 2014.

[15] Y. Dhebar and K. Deb, "A computationally fast multimodal optimization with push enabled genetic algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 191–192, ACM, 2017.

[16] Y. Dhebar and K. Deb, "Effect of a push operator in genetic algorithms for multimodal optimization," in *International Conference on Computational Intelligence, Communications, and Business Analytics*, pp. 3–21, Springer, 2017.

[17] A.-R. Hedar, "Test problems for constrained global optimization." http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page422.htm, 2018.

[18] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer methods in applied mechanics and engineering*, vol. 186, no. 2, pp. 311–338, 2000.