Extreme Reactive Portfolio (XRP): Tuning an Algorithm Population for Global Optimization

Mauro Brunato and Roberto Battiti

Dept. of Computer Science and Telecommunications, University of Trento, Italy
{brunato,battiti}@disi.unitn.it

Abstract. Given the current glut of heuristic algorithms for the optimization of continuous functions, in some case characterized by complex schemes with parameters to be hand-tuned, it is an interesting research issue to assess whether competitive performance can be obtained by relying less on expert developers (whose intelligence can be a critical component of the success) and more on automated self-tuning schemes.
After a preliminary investigation about the applicability of record statistics, this paper proposes a fast reactive algorithm portfolio based on simple performance indicators: record value and iterations elapsed from the last record. The two indicators are used for a combined ranking and a stochastic replacement of the worst-performing members with a new searcher with random parameters or a perturbed version of a well-performing member.
The results on benchmark functions demonstrate a performance equivalent or better than that obtained by offline tuning schemes, which require a greater amount of CPU time and cannot take care of individual structural variations between different problem instances.

1 Introduction: portfolios and racing for online tuning

The design of methods with complicated relationships between the algorithmic building blocks suffers from a serious illness: it hides the relevance of the various blocks, and the abundance of hand-tuned parameters impedes an objective and scientific judgment. In many cases the role of a motivated researcher, with his brain in the loop, is critical to obtain positive results [5]. A kind of dangerous data-mining exists when using a fixed set of benchmark instances and experimenting ("interrogating the data") until the tuned algorithm "screams out" acceptable results [2]. Algorithmic self-tuning has been proposed as a method for mitigating this problem. A well-known case in which a form of "online" learning mechanism is active during the search assumes that the functions to minimize satisfies some statistical model. In this manner one can develop theoretically justified methods to generate new sample points based on information derived from previous samples [21]. Another notable example in the context of Lipschitz optimization algorithms is [17].

The *algorithm portfolio* method [15] runs more algorithms concurrently, in a time-sharing manner, by allocating a fraction of the total resources to each

of them. Dynamic strategies for controlling portfolios are considered in [9, 10]. Statistical models of the quality of solutions generated by each algorithm are computed online and used as a control strategy, to determine how many cycles to allocate to each of the interleaved search strategies. A "life-long learning" approach for dynamic algorithm portfolios is considered in [13]. In [14, 16] it is shown how to use features capturing the state of a solver during the initial phase of the run to predict the length of a run, to be used by dynamic restart policies.

A related strategy to optimize the allocation of time among a set of alternative algorithms for solving a specific instance is *racing*. Running algorithms are like horses: after the competition is started one gets more and more information about the relative performance and periodically updates the bets on the winning horses, which are assigned a growing fraction of the available future computing cycles. A racing strategy is characterized by two components: i) the estimate of the future potential given the current state of the search, ii) the subsequent allocation of hardware resources to speedup the overall minimization.

Racing is related to the *k-armed bandit problem*. One is faced with a slot machine with k arms which, when pulled, yield a payoff from a fixed but unknown distribution. One wants to maximize the expected total payoff over a sequence of n trials. If the distribution is known one would immediately pull only the best performing arm. What makes the problems intriguing is that one has to split the effort between *exploration* to learn the different distributions and *exploitation* to pull the best arm, once the winner becomes clear. One is reminded of the critical exploration-versus-exploitation dilemma observed in optimization heuristics, but there is an important difference: in optimization one is not interested in maximizing the total payoff but in maximizing *the best pull* (the maximum value obtained by a pull in the sequence). The paper [12] is dedicated to determining a sufficient number of pulls to select with a high probability a hypothesis whose *average* payoff is near-optimal. The max version of the bandit problem is considered in [11, 10]. An asymptotically optimal algorithm is presented in [19], in the assumption of a generalized extreme value (GEV) payoff distribution for each arm.

When applied offline, racing algorithms search the parameter space by repeated executions of the underlying algorithm by mixing intensification and diversification phases [7]. Online racing algorithms, on the other hand, aim at performing parameter tuning during the optimization of a single instance; the goal is not to find the parametric configuration with the average best performance, but to single out good and bad runs, deciding on the spot when to replace any of them.

A way to estimate the potential of different algorithms is to put a threshold, and to estimate the probability that each algorithm produces a value above threshold by the corresponding empirical frequency. Unfortunately, the appropriate threshold is not known at the beginning, and one may end up with a trivial threshold - so that all algorithms become indistinguishable - or with an impossible threshold, so that no algorithm will reach it. The heuristic solution presented in [20] reactively learns the appropriate threshold. In spite of heuris-

tics, the specific setting of the threshold is not clear and it looks like more work is needed.

The online racing and portfolios paradigms allow to implement a global optimization metaheuristic scheme by completely *decoupling* the underlying, fixed-meta-parameter searchers from the overall parameter-tuning heuristic. In the following, we assume that a limited amount of information is periodically provided by each portfolio member, and the portfolio can be dynamically tuned while running multiple members in time-sharing, or taking advantage of multi-core or multi-machine parallelism. Dynamic tuning consists of killing underperforming members and spawning new ones with given parameter values. For simplicity of implementation we do not consider here periodic dynamic re-allocation of hardware resources.

2 The XRP reactive portfolio

Consider a pool of searchers, with different parameters. Each searcher is periodically evaluated on the basis of its past performance. Since the runtime of each searcher may vary due to machine resource contention, decisions are based on performance with respect to the number of function evaluations (optimization steps). Given the online nature of the algorithm, a member which obtained good results might be preferred to (or run alongside) a more promising member whose results are still bad due to its later start. This form of exploration vs. exploitation balance must be carefully considered when planning a portfolio strategy aimed at making sound decisions about the future of each searcher.

Depending on the goal of the optimization strategy, both short- and long-term estimates may be necessary. If running on a bounded time budget, repeated checks of short-term predictions can be necessary in order to waste as few resources as possible on underperforming searchers. On the other hand, if resources are unlimited (i.e., an *anytime* scenario in which we want to achieve better and better optima while the search advances) then longer-term predictions can be more effective by giving to every searcher enough time to smooth out their short-term behavior.

In this paper we will focus on short-term strategies where the portfolio algorithm will decide about which member algorithm to continue according to the member's performance after a given time interval.

2.1 Prediction of "time before next record"

A fundamental building block of a portfolio selection procedure is the estimation of the time before the next improvement of a local searcher, so that runs for which improvements are not expected for a long time can be stopped in favor of more promising ones.

To evaluate whether such estimate can be applied, it is possible to resort to the extreme value statistic theory. Let $y_1 = f(\boldsymbol{x}_1), y_2 = f(\boldsymbol{x}_2), \ldots$ be the values of the target function along the search trajectory of an optimization algorithm.

Let $L(n)$ be the iteration at which the nth record value is achieved. Clearly, $L(1) = 1$ (the first record is achieved at the first evaluation), while $L(n) = \min\{i | y_i < y_{L(n-1)}\}$ (the first evaluation at which the target value falls below the previous record).

A classical result [21] is that, if the y_i's are i.i.d. random variables (i.e., a searcher that evaluates a new random point at each iteration), and if N evaluations have already been performed, then the probability that the next record is achieved at iteration $N' > N$ does not depend on the past history or on the particular distribution the y_is are drawn from:

$$\Pr(L(n+1) = N' | L(n) \leq N) = \frac{N}{N'(N'-1)}. \tag{1}$$

Therefore, the probability that a new record is achieved *within* iteration N' (the corresponding c.d.f.) is also independent on the distribution and is given by

$$\Pr(L(n+1) \leq N' | L(n) \leq N) = \sum_{j=N+1}^{N'} \frac{N}{N'(N'-1)} = 1 - \frac{N}{N'} \tag{2}$$

The probability distribution is heavy-tailed, and the expected number of iterations before a new record is infinite. To define a viable criterion to model the time of the next record, it is possible to set a fixed probability value p and define the upper bound N'_p within which a new record will be achieved with probability p:

$$N'_p = \frac{N}{1-p}. \tag{3}$$

A simple experimental procedure can verify to what extent an actual search algorithm, whose y_is are not i.i.d., can be modeled by Eq. (2). Fig. 1 (left) shows the comparison between the estimated and the actual probability of a record value being achieved in a specific interval by an optimization run. After fixing a target number of function evaluations $N = 100, 1000, 10000, 100000$ (see the legend) and a target probability value $0 \leq p < 1$ (horizontal axis), the upper bound N'_p is computed and the probability tested by running 1000 optimization runs and counting how many of them achieve a new record value between iterations N and N'_p. The estimated probability (vertical axis) is plotted against the target probability with a 95% confidence interval. We can observe that the i.i.d. hypothesis systematically underestimates the probability of achieving a record during a given interval with respect to the local searcher's behavior. This is expected, because subsequent values of the search are strongly dependent, and new records tend therefore to appear in bursts which are not considered in the i.i.d. hypothesis.

To mitigate the effect of record bursts caused by the dependence of subsequent evaluations, we may want to consider a whole record burst as a single record. To identify a burst, let us fix a small probability value $q = .01$; when a new record value y_N is achieved at iteration N, let us compute the target iteration N'_q; if a new record value is achieved within iteration N'_q, then we can

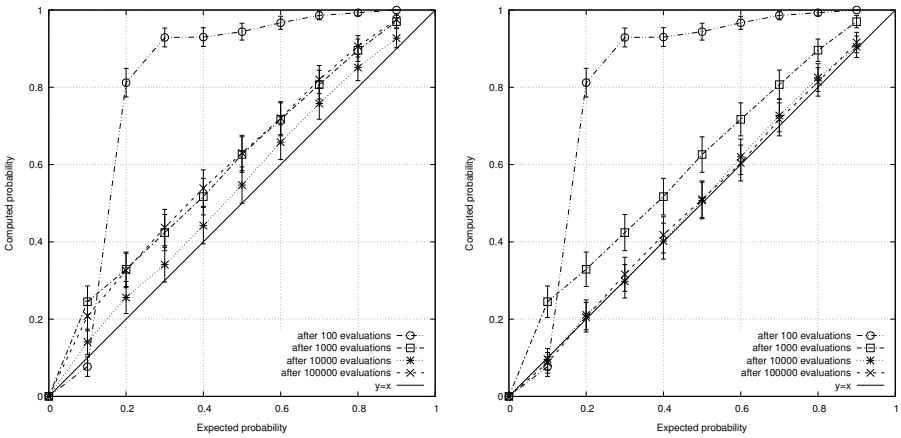


Fig. 1. Left: comparison between the expected probability of a new record event under the i.i.d. hypothesis and the actual probability computed on a sample of 1000 Inertial Shaker runs (see Sec. 3) on the 10-dimensional Rosenbrock function. Right: same comparison after collapsing record bursts.

say (with $1 - q = 99\%$ confidence) that it is not independent from y_N. By iterating this procedure, we identify a sequence of records, each being dependent on the previous one with high confidence. Let us define this sequence a *burst*, and only keep the initial (or final) iteration of a burst as the actual record. With this normalization, the correspondence between the target probability and its estimate, shown in Fig. 1 (right), is well within the 95% confidence interval for $N \geq 10000$.

This observation suggests that, while the i.i.d. hypothesis is too strong at the single function evaluation level, it holds with reasonable confidence at the burst level. In other words, the probability of a new record within a fixed number of iterations only depends on the number of iterations performed, and not on the past history of the search algorithm, *unless a record burst is currently taking place*, in which case a new record is more likely.

Based on the observations above, the evaluation of an algorithm's likelihood to produce a record in the near future will only be based on the number of iterations elapsed since the last record (as an indication of the run's likelihood to be in a dependence-fueled burst of records), without further consideration of the past history.

2.2 The Extreme Reactive Portfolio Algorithm

The Extreme Reactive Portfolio (XRP) procedure is outlined in Fig. 2. XRP works by maintaining and concurrently running a population $\mathcal{P}$ of searchers by removing underperforming ones and replacing them with new ones.

Function parameters	
f	Function to minimize
$\mathcal{A}$	Set of parametric search algorithm
$n_{\text{instances}}$	Number of concurrent instances
Δt	Time interval between population changes in seconds
p_{rnd}	Probability of replacement vs. perturbation
ϕ_{perturb}	Parameter perturbation amount
Local variables	
$\mathcal{P}$	Population of currently running search instances
$\mathcal{R}_{\text{combined}}$	Search instances sorted from worst to best combined performance
$\mathcal{R}_{\text{record}}$	Search instances sorted from best to worst record value

```
1.  function XRP (f, A, n_instances, Δt, p_rnd, φ_perturb)
2.      P ← population of n_instances random solvers from A
3.      while computational budget available
4.          Store progress information from running instances
5.          With Δt periodicity
6.              R_combined ← worst_to_best( P)
7.              bad_instance ← select( R_combined)
8.              with probability p_rnd
9.                  new_instance ← new random instance from A
10.             else
11.                 R_record ← best_to_worst( P \ {bad_instance})
12.                 good_instance ← select( R_record)
13.                 new_instance ← perturb(good_instance, φ_perturb)
14.             P ← P \ {bad_instance} ∪ {new_instance}
15.     return best configuration in P
```

Fig. 2. The Extreme Reactive Portfolio (XRP) algorithm.

The procedure receives a set of n_{alg} parametric search algorithms $\mathcal{A} = \{a_1, \ldots, a_{n_{\text{alg}}}\}$. A generic search instance is given by $a_i(\alpha_{i,1}, \ldots, \alpha_{i,n_i})$, where algorithm a_i depends on n_i parameters. We will assume that the generic parameter α_{ij} (the j-th parameter of algorithm i) has a continuous distribution in a specified range of variability; however, given the nature of many search parameters, the distribution is not necessarily uniform. Therefore, each parameter will be defined by a generator function applied to a uniformly distributed parameter:

$$\alpha_{ij} = g_{ij}(u) \qquad \text{for} \quad u \in [\min_{ij}, \max_{ij}]. \tag{4}$$

For instance, a parameter that needs to vary in $[10^{-5}, 10^{-1}]$ in logarithmic scale may be described by setting $\min_{ij} = -5$, $\max_i j = -1$ and $g_{ij}(u) = 10^u$.

XRP maintains a population $\mathcal{P}$ of running instances initialized at line 2, with each running instances being the instantiation of an algorithm randomly chosen from $\mathcal{A}$ with random parameters. The number of running instances is provided as a metaparameter; a possible choice can be given by the level of parallelism of the problem (e.g., the number of CPU cores in the machine if the evaluations

do not depend on resources shared between cores), but in principle any number of instances is admissible.

In the main loop (lines 3–14) XRP maintains a history of each instance based on the instance reports, stored as a list of triplets (current record, number of evaluations performed, elapsed CPU time). Periodically, every Δt seconds, a change is performed in the running population by selecting a bad search instance for termination (lines 6–7) and replacing it with either a new random search instance (line 9) or the perturbed version of a well-performing instance (lines 11–13). The choice of the instance to be removed is driven by a combination of two factors: the record value, and the number of function evaluations since the last improvement. In particular:

- in line 6, the running instances are ordered from worst to best performing according to a performance index that depends on the record value and the number of evaluations elapsed since the last improvement. The performance index must be independent from the relative magnitudes of the two stated factors, therefore everything is computed on the basis of ranking. Let r_i^{record} be the ranking of instance i within $\mathcal{P}$ with respect to its record value (smallest first); let r_i^{fast} be the ranking of instance i with respect to the evaluations elapsed since the last improvement (smallest first). Then, function `worst_to_best`($\mathcal{P}$) sorts instances according to their combined rankings $r_i^{\text{record}} + r_i^{\text{fast}}$ in decreasing order, so that instances with a bad performance with respect to at least one criterion move up the final ranking.
- in line 7, the bad instance to be removed is selected. To improve differentiation, the worst instance (first in ranking) isn't always chosen; rather, the choice procedure is randomized by selecting the first instance in the given ranking with probability 1/2, and every subsequent instance with half the residual probability, with the exception of the best instance (the last in the ranking) which is never selected, hence implementing a form of elitism.

After the selection, *bad_instance* will be replaced with a new instance, starting from a new random point, with two possible choices: the replication and perturbation of a well-performing instance, so as to increase the population density in proximity of good performers (intensification), or the creation of new random instance (diversification):

- With probability p_{rnd} a random instance is created (line 9).
- With probability $1 - p_{\text{rnd}}$, a running instance (different from the bad one) is selected with a probability that depends on its ranking with respect to the record value. In detail, function `best_to_worst` sorts all instances by increasing record value giving rank $\mathcal{R}_{\text{record}}$ which maps instance i to ranking r_i^{record} (line 11); an instance, called *good_selection*, is selected with the same random mechanism described above (function `select`, line 12) and a new instance is created by perturbing its parameters by a small amount (line 13).

Given a generic parameter $\alpha_{ij} = g_{ij}(u)$, in the notation of Eq. (4), the perturbed parameter is $\alpha'_{ij} = g_{ij}(u')$, where u' is a perturbed version of u obtained

by using ϕ_{perturb} on u as a relative perturbation and clamping the result within the admissible range: $u' = \text{clamp}\big(u \cdot \text{rnd}(1 \pm \phi_{\text{perturb}}), [\min_{ij}, \max_{ij}]\big)$.

3 The competing algorithms

The portfolio of competing algorithms is composed of two stochastic local search methods (RAS and IS), with a different adaptation of the dynamic local search area, and of a more complex global optimization scheme (CRTS) combining local search with prohibition-based diversification. No explicit restart is needed for the local search streams because of the online substitution of under-performing or stuck runs considered in our specific proposal.

The following sections briefly outline each technique and describe the parameters that they expose towards the portfolio algorithm.

Local Search with the Reactive Affine Shaker (RAS) The Reactive Affine Shaker Heuristic [8] (RAS) is a self-tuning local search algorithm which does not assume prior knowledge on the function to be minimized. The function is considered as a "black box" (oracle) which can be interrogated to get output values corresponding to input values. The RAS heuristic tries to rapidly move towards better objective values by maintaining and reshaping a bounded *search region* $\mathcal{S}$ around the current point $\boldsymbol{x}$.

The search region is reshaped on the basis of success (or lack of success) during the last step: if a step in a certain direction yields a better objective value, then $\mathcal{S}$ is expanded along that direction; it is contracted otherwise. Therefore, once a promising direction is found, the probability that subsequent steps will follow the same direction is increased, and the search shall proceed more and more aggressively in that direction until bad results reduce its prevalence.

The RAS heuristic depends on two parameters, which are exposed to XRP: an initial box width coefficient $0 < \eta \leq 1$, defining the size of the initial, isotropic search region with respect to the size of the function domain; and a contraction coefficient $0 < f_{\text{con}} \leq 1$ governing the contraction of the search region along unsuccessful directions; a corresponding expansion coefficient for successful search directions is obtained automatically as $f_{\text{dil}} = f_{\text{con}}^{-1}$.

Local Search with the Inertial Shaker (IS) RAS requires matrix-vector multiplications to update the search region, and therefore slows down when the number of dimensions becomes very large. The simpler *Inertial Shaker* (IS) technique [3] can be a more effective choice in this case: the search box is always identified by vectors parallel to the coordinate axes (therefore the search box is defined by a single vector $\boldsymbol{\beta}$ and no matrix multiplications are needed) and a *trend direction* is identified by averaging the d previous displacements where d is the domain's dimensionality.

The IS heuristic exposes one parameter to XRP: the amplification coefficient, defined as $f_{\text{ampl}} > 0$, controlling the extent at which the trend direction mentioned above modifies the search box.

Continuous Reactive Tabu Search for Global Optimization While the two previous heuristics do not contain an explicit diversification mechanism, and are therefore local search schemes, the *Continuous Reactive Tabu Search* heuristic [4] (CRTS) can use any of them as a basic searcher while incorporating elements to achieve global optimization.

The initial search region, specified by bounds on each independent variable, is recursively partitioned into a *tree of boxes* (with axes parallel to the coordinate axes). The tree is born with 2^d equally sized leaves, obtained by dividing in half the initial range on each variable. Each box is then subdivided into 2^d equally-sized children, as soon as two different local minima are found in it. The leaves of the tree partition the domain and are the admissible starting regions for the combinatorial component of CRTS.

While the underlying local search algorithm generates a search trajectory consisting of points $X^{(t)}$, CRTS maintains a trajectory consisting of *leaf-boxes* that are evaluated by sampling.

Moving from a leaf-box to another is done on the basis of their neighborhood, their sampled value and a prohibition list whose size is automatically adjusted depending on the search success.

The CRTS heuristic exposes one parameter to XRP: the tabu list size reduction factor $0 < f_{\text{red}} \leq 1$, controlling the extent by which the prohibition list is reduced in presence of successful results. The corresponding expansion factor is determined as $f_{\text{exp}} = f_{\text{red}}^{-1}$.

CRTS also exposes to XRP the f_{con} parameter if used in collaboration with RAS (CRTS+RAS) (η is automatically set by CRTS), and the f_{ampl} parameter if used in conjunction with IS (CRTS+IS).

4 Experimental results

We tested XRP on various classical benchmark functions. The Rosenbrock, Sphere and Zakharov function families are unimodal, while Goldstein-Price, Hartmann, Rastrigin, and Shekel are multi-modal.

XRP was tested on an 8-core 2.33GHz Intel Xeon server running a 64-bit Linux OS with kernel 3.13.0. The basic search algorithms (RAS, Inertial Shaker, CRTS), were implemented in C++, while the portfolio selection code was written in Python 2.7. The instances were executed as separate processes, reporting newly found minima via standard output.
Instances were created by randomly selecting a heuristic from the set $\mathcal{A} = \{a_1, \ldots, a_4\}$ of the four heuristics described in Section 3:

- a_1 is the Reactive Affine Shaker (RAS) with $n_1 = 2$ free parameters:
  (i) initial box width coefficient $\eta = \alpha_{11} = 10^{-u}$, u uniformly selected in $[\min_{11}, \max_{11}] = [1, 3]$;
  (ii) contraction coefficient $f_{\text{con}} = \alpha_{12} = 1 - u^3$, u uniformly selected in $[\min_{12}, \max_{12}] = [.01, .7]$.
  This choice of generator functions favors contraction coefficients close to 1, as small changes in the box size need to be more finely tuned with respect to

large ones, and treats the initial box width as uniform in logarithmic values. The dilation coefficient is automatically determined as $f_{\text{dil}} = f_{\text{con}}^{-1}$.

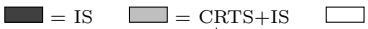
- a_2 is the Inertial Shaker (IS) with $n_2 = 1$ free parameter, the amplification coefficient, defined as $f_{\text{ampl}} = \alpha_{21} = 1 - u^3$, with u uniformly selected in $[\min_{21}, \max_{21}] = [.01, .7]$.
  The rationale for the choice of the generator function is the same as in the RAS case.
- a_3 is CRTS with RAS as local searcher, with $n_3 = 2$ free parameters:
  (i) the prohibition reduction factor $f_{\text{red}} = \alpha_{31}$ is chosen uniformly in $[\min_{31}, \max_{31}] = [.5, 1]$;
  (ii) RAS's contraction coefficient $f_{\text{con}} = \alpha_{32}$ is defined as in a_1.
- a_4 is CRTS with IS as local searcher, with $n_4 = 2$ free parameters:
  (i) the prohibition reduction factor $f_{\text{red}} = \alpha_{41}$ is defined as in a_3;
  (ii) the amplification coefficient $f_{\text{ampl}} = \alpha_{42}$ is defined as in a_2.

Metaparameters determination As described in Section 2.2, XRP depends on a number of metaparameters. The number of concurrent instances has been set to the number of cores in the experimental server, $n_{\text{instances}} = 8$. The remaining three metaparameters have been set by experimenting on 60-second optimizations of the 80-dimensional Rastrigin function. The function was chosen because it is barely solvable within the allocated time limit, with many search instances being unable to find the minimum, and it wasn't used in the following assessment (only lower-dimensional instances will).

- Time interval Δt between two consecutive instance replacement events. Experiments show that a small time interval ($\Delta t \leq 2$s) imposes a significant startup overhead on the operating system, so that some processes failed to start within the next period. Tests using $\Delta t = 2.5\text{s}, 5\text{s}, 10\text{s}, 20\text{s}$ appear to place the best value in the interval $[2.5\text{s}, 10\text{s}]$; setting $\Delta t = 20$s shows a performance deterioration because of the small instance turnover in the allotted minute. The selected value for subsequent tests is $\Delta t = 5$s.
- Probability of random replacement p_{rnd} governing the choice between a new random instance and a perturbed version of a running instance. Values of $p_{\text{rnd}} = 0, .25, .5, .75, 1$ have been tested. Tests on the extreme values ($p_{\text{rnd}} = 0$ and $p_{\text{rnd}} = 1$) show that many runs (20 of 30) do not converge to the minimum (due to excessive focus on some instances or, conversely, to excessive randomness). The central values seem to be approximately equivalent, so $p_{\text{rnd}} = .5$ has been selected.
- Relative parameter perturbation factor ϕ_{perturb}. Values of 0, .1 and .2 have been tested; as expected, for $\phi_{\text{perturb}} = 0$ a fraction of the runs (17 of 30) do not converge to the minimum, probably because of too little differentiation between good instances. We chose $\phi_{\text{perturb}} = .1$.

In general, we observe that XRP is quite robust with respect to metaparameter changes, provided that they fall within a reasonable range, therefore no further optimization has been attempted beyond this preliminary investigation.

Table 1. Median record values for 30 runs of XRP, best algorithm and iRace-determined algorithm for 10^7 evaluations of the target function. The gray bars show the distribution of the winner algorithm over the 30 runs.



Function	Dim	XRP Median	XRP IQR	Winner Median	Winner IQR	iRace winner Median	iRace winner IQR
Goldstein -Price	2	$5.62{\cdot}10^{-12}$	$1.85{\cdot}10^{-11}$	$1.77{\cdot}10^{-12}$	$1.47{\cdot}10^{-11}$	$7.89{\cdot}10^{-05}$	$1.27{\cdot}10^{-04}$
Hartmann	3	$1.24{\cdot}10^{-14}$	$5.51{\cdot}10^{-14}$	$6.66{\cdot}10^{-15}$	$2.89{\cdot}10^{-14}$	$3.52{\cdot}10^{-06}$	$3.67{\cdot}10^{-06}$
Hartmann	6	$1.33{\cdot}10^{-12}$	$2.59{\cdot}10^{-12}$	$5.44{\cdot}10^{-13}$	$1.99{\cdot}10^{-12}$	$5.05{\cdot}10^{-05}$	$4.53{\cdot}10^{-06}$
Rastrigin	10	$1.10{\cdot}10^{-08}$	$2.39{\cdot}10^{-03}$	$4.19{\cdot}10^{-09}$	$6.28{\cdot}10^{-04}$	$2.63{\cdot}10^{-02}$	$2.02{\cdot}10^{-02}$
Rastrigin	30	$2.01{\cdot}10^{-13}$	$1.55{\cdot}10^{-12}$	$8.53{\cdot}10^{-14}$	$2.39{\cdot}10^{-13}$	$6.06{\cdot}10^{-13}$	$2.14{\cdot}10^{-09}$
Rosenbrock	10	$3.35{\cdot}10^{-13}$	$3.29{\cdot}10^{-10}$	$5.38{\cdot}10^{-20}$	$1.40{\cdot}10^{-16}$	$7.82{\cdot}10^{-19}$	$4.86{\cdot}10^{-16}$
Rosenbrock	30	$1.72{\cdot}10^{-14}$	$3.99{\cdot}10^{-14}$	$6.73{\cdot}10^{-15}$	$7.91{\cdot}10^{-15}$	$6.99{\cdot}10^{-15}$	$3.75{\cdot}10^{-14}$
Shekel 5	4	$2.53{\cdot}10^{-10}$	$8.16{\cdot}10^{-10}$	$6.49{\cdot}10^{-11}$	$2.08{\cdot}10^{-10}$	$2.17{\cdot}10^{-02}$	$1.84{\cdot}10^{-02}$
Shekel 7	4	$1.87{\cdot}10^{-10}$	$5.98{\cdot}10^{-10}$	$4.49{\cdot}10^{-11}$	$1.53{\cdot}10^{-10}$	$2.19{\cdot}10^{-02}$	$1.58{\cdot}10^{-02}$
Shekel 10	4	$1.16{\cdot}10^{-10}$	$4.53{\cdot}10^{-10}$	$3.86{\cdot}10^{-11}$	$3.72{\cdot}10^{-10}$	$3.06{\cdot}10^{-02}$	$2.03{\cdot}10^{-02}$
Sphere	30	$3.78{\cdot}10^{-79}$	$7.31{\cdot}10^{-77}$	$1.47{\cdot}10^{-90}$	$1.55{\cdot}10^{-83}$	$2.35{\cdot}10^{-87}$	$4.11{\cdot}10^{-80}$
Zakharov	10	$8.50{\cdot}10^{-69}$	$1.55{\cdot}10^{-62}$	$4.74{\cdot}10^{-82}$	$9.85{\cdot}10^{-74}$	$1.72{\cdot}10^{-82}$	$1.31{\cdot}10^{-72}$
Zakharov	30	$3.95{\cdot}10^{-81}$	$7.96{\cdot}10^{-70}$	$3.38{\cdot}10^{-319}$	$2.39{\cdot}10^{-319}$	$2.98{\cdot}10^{-319}$	$1.67{\cdot}10^{-319}$

4.1 Evaluation with a fixed budget

For each of the benchmark functions listed in Table 1, 30 runs of XRP were performed with a total budget of 10^7 evaluations. The median and inter-quartile range of all record results are reported in the "XRP" column. Results are reported as the difference from the actual global minimum value (Goldstein-Price, Hartmann and Shekel families have non-zero global minima). The "XRP" column also shows the distribution of the winner algorithm over the 30 runs of XRP for each benchmark function (the search instance that achieved the record value over the portfolio selection run).

It is possible to observe that the majority of winners involves the simpler IS technique, with the RAS algorithm only appearing on two runs, both times in conjunction with CRTS. As expected, many low-dimensional instances ($d \leq 10$) benefit from the diversification capabilities of CRTS; the exceptions are the unimodal Rosenbrock and Zakharov instances, where a single local search run is sufficient, and the four-dimensional Shekel instances, whose few foxhole-shaped local minima trick the CRTS district evaluation mechanism, but are few (resp. 5, 7, and 10) and narrow enough for a well-tuned local searcher to jump in and out of them without the need to restart. High-dimensional instances ($d = 30$), on the other hand, the CRTS technique doesn't have the time to start

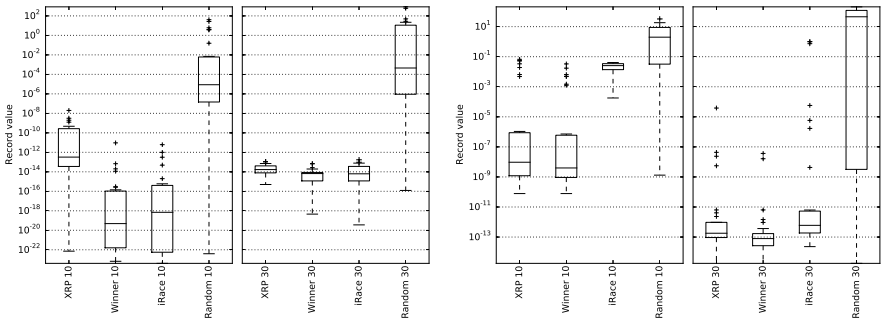


Fig. 3. Distribution of record values after 10^6 function evaluations for XRP, Winner, iRace-determined and random heuristic for the 10- and 30-dimensional Rosenbrock (left) and Rastrigin (right) functions.

evaluating all first-level districts (2^{30}), while the local searchers don't suffer from that disadvantage.

After each run, the search algorithm instance that achieved the record result was allocated the whole 10^7 evaluations budget for a separate run with the same random seed, so that its execution would exactly mimic its run within XRP, but for a longer time. The 30-test median and IQR are reported in column "Winner".

Results in column "iRace" will be discussed in Sec. 4.3. Observe that in most cases the XRP outcome is quite close to the "Winner" result, within the same order of magnitude. Notable exceptions are the simplest unimodal Sphere function, quadratic with spherical symmetry, and Zakharov's very flat global attractor, whose small final improvements tend to require quite long runs. Even in these cases, however, the XRP outcome is well within reasonable target limits.

The outcomes for the 10- and 30-dimensional Rosenbrock and Rastrigin functions are also reported in Fig. 3 for ease of comparison. Observe that, in some cases, the 30-dimensional problem is solved more efficiently than the 10-dimensional one, in particular for the Rastrigin function. When evaluating the 10-dimensional function, bad optimization instances can consume a large part of the budget before XRP can check and eventually stop them. This effect is mitigated by the longer time needed to evaluate the 30-dimensional version. Finer control over the instance performance, overcoming the strict periodicity of the current algorithm, will possibly remove this kind of artifact.

The "Random" box refers to a series of 30 runs of random heuristics with randomly determined parameters. Note that, in all cases, the choice of random instances achieves much worse results; thus motivating the use of online portfolio selection when no prior information about the optimal algorithm and parameter settings is given.

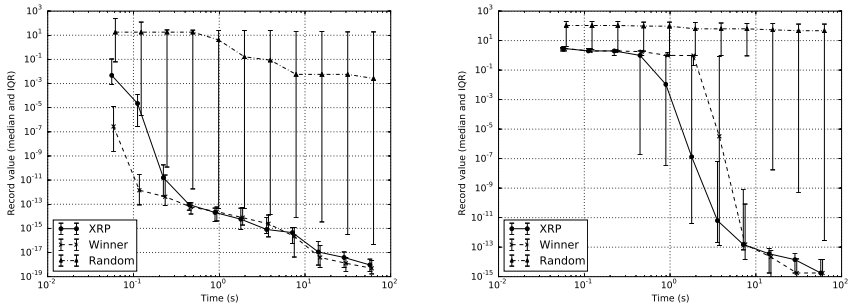


Fig. 4. Median and IQR of record values for 30 60-second runs of XRP, Winner heuristic and Random heuristic, for 30-dimensional Rosenbrock (left) and 30-dimensional Rastrigin (right). Note: inter-quartile bars have been slightly offset to tell them apart in spite of overlaps.

4.2 Evolution of record values in time

Fig. 4 shows the evolution of the record value over 60 seconds of XRP, comparing it with the best instance running for the same time and with the same seed (see Sec. 4.1) and with a random instance. Curves represent median values of 30 runs, with bars representing the interval between the 1st and 3rd quartiles.

The overlap between the "XRP" and "Winner" IQR bars shows that many runs fall in the same range; while in the end the Winner instance always wins, for a short initial time (consider that Fig. 4 is a log-log plot) the overall XRP outcome can outperform the Winner alone due to the concurrent execution of many instances, in particular when a more difficult function such as Rastrigin is chosen (with its many local minima). Again, a random choice is almost always penalized.

4.3 Comparison with offline racing

To identify a good algorithm/parameter combination that would work over the whole set of benchmark functions, we trained the offline racing package iRace [18] on a budget of 1000 60-second optimization runs over the whole function set, with the same parameter distribution as the online procedure. The Inertial Shaker (IS) algorithm with amplification coefficient $f_{\text{ampl}} = .7033$ was identified as the best tradeoff.

The last column of Table 1 shows the distribution of the record value for 30 10^7-iteration runs of the optimal algorithm on each benchmark function. We can observe that the resulting parameter set was not equally good for all functions; in particular, it performs very well on Zakharov, Sphere, and the 30-dimensional Rosenbrock and Rastrigin functions (also see Fig. 3 for a visual comparison of result distributions), while it seems to work less than optimally for other instances. This result confirms the heuristic observation that parameter tuning

is needed at the instance level, and supports our rationale for online tuning (when the anytime nature of the solution method allows for it).

5 Conclusions

A novel online dynamic portfolio, XRP, for anytime optimization heuristics has been presented and tested on classical function minimization benchmarks. XRP can be applied to any collection of optimization heuristics with the only requirement to be able to report their record values as soon as they are achieved. While XRP itself depends on few simple metaparameters, preliminary investigation indicates that results are robust with respect to variations from their natural central values.

Further investigation will consider the use of XRP for more challenging very large-scale optimization problems, both continuous and discrete, and the implementation of more sophisticated models for the performance of the competing instances, in order to reliably identify unpromising executions and their most suitable replacements as optimization proceeds. An evaluation of schemes based on reinforcement learning [6, 1] is also on the stack.

Acknowledgments

The research of Roberto Battiti was supported by the Russian Science Foundation, project no. 15–11–30022 "Global optimization, supercomputing computations, and applications."

References

1. Battiti, R., Brunato, M., Campigotto, P.: Learning while optimizing an unknown fitness surface. In: Maniezzo, V., Battiti, R., Watson, J.P. (eds.) LION II: Learning and Intelligent OptimizatioN Conference, Trento, Italy, Dec 10-12, 2007. Lecture Notes in Computer Science, vol. 5313. Springer Verlag (2008)
2. Battiti, R., Brunato, M., Mascia, F.: Reactive Search and Intelligent Optimization, Operations research/Computer Science Interfaces, vol. 45. Springer Verlag (2008)
3. Battiti, R., Tecchiolli, G.: Learning with first, second, and no derivatives: a case study in high energy physics. Neurocomputing 6, 181–206 (1994)
4. Battiti, R., Tecchiolli, G.: The continuous reactive tabu search: blending combinatorial optimization and stochastic search for global optimization. Annals of Operations Research – Metaheuristics in Combinatorial Optimization 63, 153–188 (1996)
5. Battiti, R., Brunato, M.: The LION way. Machine Learning *plus* Intelligent Optimization. LIONlab, University of Trento, Italy (2014)
6. Battiti, R., Campigotto, P.: Reinforcement learning and reactive search: an adaptive max-sat solver. In: M. Ghallab, C.D. Spyropoulos, N.F., Avouris, N. (eds.) Proceedings ECAI 08: 18th European Conference on Artificial Intelligence, Patras, Greece, Jul 21-25, 2008. IOS Press, Amsterdam (2008)

7. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: *et al.*, W.L. (ed.) Proceedings of the Genetic and Evolutionary Computation Conference. pp. 11–18. Morgan Kaufmann Publishers, San Francisco, CA, USA (2002), also available as: AIDA-2002-01 Technical Report of Intellektik, Technische Universität Darmstadt, Darmstadt, Germany
8. Brunato, M., Battiti, R., Pasupuleti, S.: A memory-based rash optimizer. In: Geffner, A.F.R.H.H. (ed.) Proceedings of AAAI-06 workshop on Heuristic Search, Memory Based Heuristics and Their applications. pp. 45–51. Boston, Mass. (2006), iSBN 978-1-57735-290-7
9. Cicirello, V.: Boosting Stochastic Problem Solvers Through Online Self-Analysis of Performance. Ph.D. thesis, Carnegie Mellon University, Also available as technical report CMU-RI-TR-03-27. (2003)
10. Cicirello, V., Smith, S.: The max k-armed bandit: A new model for exploration applied to search heuristic selection. In: 20th National Conference on Artificial Intelligence (AAAI-05) (July 2005)
11. Cicirello, V.A., Smith, S.F.: Principles and Practice of Constraint Programming CP 2004, Lecture Notes in Computer Science, vol. 3258, chap. Heuristic Selection for Stochastic Search Optimization: Modeling Solution Quality by Extreme Value Theory, pp. 197–211. Springer Berlin / Heidelberg (2004)
12. Fong, P.W.L.: A quantitative study of hypothesis selection. In: International Conference on Machine Learning. pp. 226–234 (1995), `citeseer.ist.psu.edu/fong95quantitative.html`
13. Gagliolo, M., Schmidhuber, J.: Learning dynamic algorithm portfolios. In: Proceedings AI and MATH '06, Ninth International Symposium on Artificial Intelligence and Mathematics. Fort Lauderdale, Florida (Jan 2006)
14. Horvitz, E., Ruan, Y., Gomes, C., Kautz, H., Selman, B., Chickering, D.M.: A bayesian approach to tackling hard computational problems. In: Seventeenth Conference on Uncertainty in Artificial Intelligence. pp. 235–244. Seattle,USA (Aug 2001)
15. Huberman, B.A., Lukose, R.M., Hogg, T.: An economics approach to hard computational problems. Science 275, 51–54 (January 3 1997)
16. Kautz, H., Horvitz, E., Ruan, Y., Gomes, C., Selman, B.: Dynamic restart policies. In: Eighteenth national conference on Artificial intelligence. pp. 674–681. American Association for Artificial Intelligence, Menlo Park, CA, USA (2002)
17. Lera, D., Sergeyev, Y.D.: Acceleration of univariate global optimization algorithms working with lipschitz functions and lipschitz first derivatives. SIAM Journal on Optimization 23(1), 508–529 (2013)
18. López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)
19. Streeter, M.J., Smith, S.F.: An asymptotically optimal algorithm for the max k-armed bandit problem. In: Proceedings of the National Conference on Artificial Intelligence. vol. 21, pp. 135–142. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999 (2006)
20. Streeter, M.J., Smith, S.F.: A simple distribution-free approach to the max k-armed bandit problem. In: Principles and Practice of Constraint Programming-CP 2006. pp. 560–574. Springer (2006)
21. Zhigljavsky, A., Žilinskas, A.: Stochastic global optimization, vol. 9. Springer Science & Business Media (2007)