Efficient Multi-start Strategies for Local Search Algorithms*

Levente Kocsis and András György

Machine Learning Research Group
Computer and Automation Research Institute
of the Hungarian Academy of Sciences
Kende u. 13-17, 1111 Budapest, Hungary
kocsis@sztaki.hu, gya@szit.bme.hu

Abstract. Local search algorithms for global optimization often suffer from getting trapped in a local optimum. The common solution for this problem is to restart the algorithm when no progress is observed. Alternatively, one can start multiple instances of a local search algorithm, and allocate computational resources (in particular, processing time) to the instances depending on their behavior. Hence, a multi-start strategy has to decide (dynamically) when to allocate additional resources to a particular instance and when to start new instances. In this paper we propose a consistent multi-start strategy that assumes a convergence rate of the local search algorithm up to an unknown constant, and in every phase gives preference to those instances that could converge to the best value for a particular range of the constant. Combined with the local search algorithm SPSA (Simultaneous Perturbation Stochastic Approximation), the strategy performs remarkably well in practice, both on synthetic tasks and on tuning the parameters of learning algorithms.

1 Introduction

Local search algorithms for global optimization often suffer from getting trapped in a local optimum. Moreover, local search algorithms that are guaranteed to converge to a global optimum under some conditions (such as Simulated Annealing or SPSA) usually converge at a very slow pace in order to provide consistency. On the other hand, if the algorithms are employed with more aggressive settings, much faster convergence to local optima is achievable, but with no guarantee to find the global optimum. The common solution to escape from a local optimum is to restart the algorithm when no progress is observed. Alternatively, one can start multiple instances of the local search algorithm, and allocate computational resources, in particular, processing time, to the instances depending on their behavior. The moment of starting an instance can vary in time, and so the number of instances can also grow over time depending on the allocation strategy.

* This research was supported in part by the Mobile Innovation Center of Hungary, the Hungarian Scientific Research Fund and the Hungarian National Office for Research and Technology (OTKA-NKTH CNK 77782), and by the PASCAL2 Network of Excellence (EC grant no. 216886).

W. Buntine et al. (Eds.): ECML PKDD 2009, Part I, LNAI 5781, pp. 705–720, 2009.

Running several instances of an algorithm or several algorithms in parallel and selecting among the algorithms have been intensively studied in the area of meta-learning [22]. The main problem here is to allocate time slices to particular algorithms with the aim of maximizing the best result returned. This allocation may depend on the intermediate performance of the algorithms.

A simplified version of this problem is the so called Maximum K-Armed Bandit [6], where at each time instance exactly one algorithm (called arm in this framework) can be run, and it is assumed that the intermediate results returned by an instance of the algorithm are drawn independently from a particular distribution. Note, however, that this assumption is quite unrealistic, and usually leads to oversimplification, as local search algorithms steadily improve their performance, and thus the distribution from which the next value is drawn differs significantly from the ones from which the values in the previous several iterations were drawn.

Nevertheless, the Maximum K-Armed Bandit problem provides a convenient framework, and is often used for the analysis of multi-start search strategies [6,7,20]. A generic algorithm for this problem is provided in [1], where the, so called, reservation price of an instance is introduced. If an instance achieves its reservation price, it is useless to select it again. The computation of the reservation price depends on a model of the algorithm that can be learned under some specific constraints. Several algorithms [6,7,20] are based on the (generalized) extreme value theory: these algorithms are similar to standard multi-armed bandit algorithms (see, e.g., [3]) except that the upper confidence bounds are derived from different distributions (e.g., from Gumbel distribution). In [21] a distribution free approach is proposed that combines a multi-armed bandit exploration strategy with a heuristic selection among the available arms. The resulting algorithm (called ThresholdAscent) appeared to have attractive practical performance compared to the ones based on extreme value theory. Finally, a natural strategy is to probe the algorithm instances for a while, estimate their future performance based on the results of this trial phase, and then use the most promising algorithm for the time remaining. Simple rules were suggested in [4] to predict the future performance of each algorithm, while [5] employs Bayesian prediction.

Another related problem is the following. Several algorithm instances are available that all produce the correct answer to a certain question if run for a sufficiently long time. The time needed for an algorithm instance to find the answer is a random quantity, and the goal is to combine the given algorithms to minimize the expected running time until the answer is found. When the distribution of the running time is known, an optimal time allocation strategy to minimize the expected running time is to perform a sequence of runs with a certain cut-off time that depends on the distribution [17]. If the distribution is unknown, a particular running time sequence can be chosen that results in an expected total running time that is only a logarithmic factor larger than the optimum achievable if the distribution is known. We note that this strategy is among the few that provide a schedule that increases the number of algorithm

instances. For the same set-up, an allocation strategy is proposed in [16] based on updating dynamically the belief over the run-time distribution. Finally, when a set of time allocation strategies are available and the optimization problem is to be solved several times, one can use the standard multi-armed bandit framework as in [9,10,11].

In this paper, we propose a new multi-start strategy, MetaMax, that assumes a convergence rate of the local search known up to an unknown constant. In every phase those instances are selected that may converge to the best value for a particular range of the constant. The selection mechanism is analogous to the DIRECT algorithm [15,8,14] for optimizing Lipschitz-functions with an unknown constant, where preference is given to rectangles that may contain the global optimum. The optimum within each rectangle is estimated in an optimistic way, and the estimate depends on the size of the rectangle. In our algorithm we use the function describing the convergence rate of the local search algorithms in a similar way as the size of the rectangles are used in the DIRECT algorithm.

The rest of the paper is organized as follows. Basic definitions and assumptions are given in Section 2. The new multi-start local search strategies of this paper, MetaMax(K) and MetaMax, are described and analyzed in Section 3: in Section 3.1 we deal with a selection mechanism among a fixed number of instances of the local search algorithm, while, in addition, a simple schedule for starting new instances is also considered in Section 3.2. Simulation results on real and synthetic data are provided in Section 4. Conclusions and possible further work are described in Section 5.

2 Preliminaries

Assume we wish to maximize a nonnegative real valued function f on the d-dimensional unit hypercube $[0,1]^d$, that is, the goal is to find a maximizer $x^* \in [0,1]^d$ such that $f(x^*) = \max_{x\in[0,1]^d} f(x)$. Without loss of generality, we assume that f is non-negative, and suppose, for simplicity, that f is continuous on $[0,1]^d$.[1] The continuity of f implies the existence of x^*, and, in particular, that f is bounded.

If the form of f is not known explicitly, search algorithms usually sample f at several locations and return an estimate of x^* and $f(x^*)$ based on these observations. There is an obvious trade-off between the number of samples used and the quality of the estimate, and the performance of any search strategy may be measured by the accuracy it achieves in estimating $f(x^*)$ under a constraint on the cost measured by the number of samples used.

Given a relatively good local search algorithm A, a general strategy for finding a good approximation of the optimum x^* is to run several instances of A initialized at different starting points and approximate $f(x^*)$ with the maximum f value observed. We concentrate on local search algorithms A defined formally by a sequence of possibly randomized sampling functions $s_n : [0,1]^{d\cdot n} \to [0,1]^d$,

[1] The results can easily be extended to (arbitrary valued) bounded piecewise continuous functions with finitely many continuous components.

$n = 1, 2, \ldots$: A samples f at locations $X_1, X_2, \ldots$ where $X_{i+1} = s_i(X_1, \ldots, X_i)$ for $i \geq 1$, and the starting point $X_1 = s_0$ is chosen uniformly at random from $[0,1]^d$; after n observations A returns the estimate of x^* and the maximum $f(x^*)$, respectively, by

$$\widehat{X}_n = \operatorname*{argmax}_{1 \leq k \leq n-1} f(X_k) \qquad \text{and} \qquad f(\widehat{X}_n).$$

It is clear that if the starting points are sampled uniformly from $[0,1]^d$ and each algorithm is evaluated at its starting point then this strategy is asymptotically consistent as the number of instances tend to infinity (as in the worst case we perform a random search that is known to converge to the maximum almost surely). On the other hand, if algorithm A has some favorable properties then it is possible to design multi-start strategies that still keep the random search based consistency, but provide much faster convergence to the optimum in terms of the number of evaluations of f.

In particular, we assume the following on the local behavior of the search algorithms:

Assumption 1. *The search algorithm A converges to a local maximum at a guaranteed rate, that is,*

(i) the limit $\lim_{t\to\infty} \widehat{X}_t$ exists and it is a deterministic function of the starting point X_1 for almost all values of X_1 with respect to the Lebesgue measure;

(ii) for any $0 < \delta < 1$, there exists a function $g_\delta(n)$ such that

$$\mathbb{P}\left(\lim_{t\to\infty} f(\widehat{X}_t) - f(\widehat{X}_n) \leq c g_\delta(n) \middle| X_1\right) \geq 1 - \delta \tag{1}$$

where $g_\delta(n)$ is a positive, monotone decreasing function of n that converges to 0 for any $0 < \delta < 1$, that is, $\lim_{n\to\infty} g_\delta(n) = 0$, and c is a positive constant that depends on f and g_δ.

In certain cases $g_\delta(n) = O(e^{-\alpha n})$ or $g_\delta(n) = O(n^{-\alpha})$ for some $\alpha > 0$, see, e.g., [12]. The constant c depends on certain characteristics of f, e.g., on the maximum local steepness, or, if f is a random function, on the variance of f (note, however, that in this paper we consider only deterministic functions). The above assumed property of the local search algorithms will be utilized in the next section to derive efficient multi-start search strategies.

3 Multi-start Search Strategies

Standard multi-start search strategies run an instance of A until it seems to converge to a location where there is no hope to beat the current best approximation of $f(x^*)$. An alternative way of using multiple instances of local search algorithms is to run all algorithms parallel, and in each round we decide which algorithms can take an extra step. This approach may be based on estimating the potential performance of a local search algorithm A based on Assumption 1.

Note that if c were known, an obvious way would be to run each instance as long as their possible performances become separated with high probability. Then we could just pick the best instance and run it until the end of our computational budget (this would be a simple adaptation of the idea of choosing the best algorithm based on a trial phase as in [4,5]). However, if c is unknown, we cannot make such separation. On the other hand, using ideas from the general methodology for Lipschitz optimization with unknown constant [15], we can get around this problem and estimate, in a certain optimistic way, the potential performance of each algorithm instance, and in each round we can step the most promising ones.

The main idea of the resulting strategy can be summarized as follows. Assume we have K instances of an algorithm A, denoted by $A_1, \ldots, A_K$ satisfying Assumption 1. Let $X_{i,n}, i = 1, \ldots, K$ denote the location at which f is sampled by A_i at the nth time it can take a sample, where $X_{i,1}$ is the starting point of A_i. The estimate of the location of the maximum by algorithm A_i after n samples is

$$\widehat{X}_{i,n} = \operatorname*{argmax}_{1 \le t \le n} f(X_{i,t})$$

and the maximum value of the function is estimated by $\hat{f}_{i,n} = f(\widehat{X}_{i,n})$. Now if A_i observes $n_{i,r}$ samples by the end of the rth round, by Assumption 1 we have, with probability at least $1 - \delta$,

$$f(\bar{X}_i) - \hat{f}_{i,n_{i,r}} \le c g_\delta(n_{i,r})$$

where $\bar{X}_i = \lim_{n\to\infty} \widehat{X}_{i,n}$ is the point where A_i converges. Thus, an optimistic estimate of the maximum of f for a constant $\hat{c}$, based on A_i, is

$$\hat{f}_{i,n_{i,r}} + \hat{c} g_\delta(n_{i,r}).$$

Then it is reasonable to choose, in each round, those algorithms to take another step that provide the largest estimate for some values of $\hat{c}$ (in this way we can get around the fact that we do not know c).

3.1 Constant Number of Instances

The above idea can be translated to the algorithm MetaMax(K) shown in Figure 1. Here we consider the case when we have a fixed number of instances, and our goal is to perform (almost) as well as the best of them (in hindsight), while using the minimum number of evaluations of f. Note the slight abuse of notation that in the MetaMax(K) algorithm $\widehat{X}_r$ and $\hat{f}_r$ denote the estimates of the algorithm after r rounds (and not r samples).

In each round, the strategy MetaMax(K) selects the local search algorithms A_i for which the point $(g_\delta(n_{i,r-1}), \hat{f}_{i,n_{i,r-1}})$ lies on the upper convex hull of the set

$$\mathcal{P}_r = \{(g_\delta(n_{j,r-1}), \hat{f}_{j,n_{j,r-1}}) : j = 1, \ldots, K\} \cup \{(0, \max_{1 \le j \le K} \hat{f}_{j,n_{j,r-1}})\}. \quad (3)$$

METAMAX(K): A MULTI-START STRATEGY WITH K ALGORITHM INSTANCES.

Parameters: $K > 0$, $0 < \delta < 1$, $g_\delta(n)$.
Initialization: For each $i = 1, \ldots, K$, let $n_{i,0} = 0$, $\hat{f}_{i,0} = 0$.

For each round $r = 1, 2, \ldots$

(a) For $i = 1, \ldots, K$ select algorithm A_i if there exists a $\hat{c} > 0$ such that

$$\hat{f}_{i,n_{i,r-1}} + \hat{c} g_\delta(n_{i,r-1}) > \hat{f}_{j,n_{j,r-1}} + \hat{c} g_\delta(n_{j,r-1}) \qquad (2)$$

for all $j \neq i$ such that $n_i \neq n_j$. If there are several values of i selected that have the same sample number $n_{i,r-1}$ then keep only one of these selected uniformly at random.

(b) Step each selected A_i, and update variables. That is, set $n_{i,r} = n_{i,r-1} + 1$ if A_i is selected, and $n_{i,r} = n_{i,r-1}$ otherwise. For each selected A_i sample $f(X_{i,n_{i,r}})$ and compute the new estimates $\widehat{X}_{i,n_{i,r}}$ and $\hat{f}_{i,n_{i,r}}$.

(c) Let $I_r = \operatorname{argmax}_{i=1,\ldots,K} \hat{f}_{i,n_{i,r}}$ denote the index of the algorithm with the currently largest estimate of $f(x^*)$, and estimate the location of the maximum with $\widehat{X}_r \widehat{X}_{I_r,n_{I_r,r}}$ and its value with $\hat{f}_r = \hat{f}_{I_r,n_{I_r,r}}$.

Fig. 1. The METAMAX(K) algorithm

Note that it is guaranteed that in each round we sample at least two algorithms, one with the largest estimate $\hat{f}_{n_{i,r-1}}$, and one with the smallest sample size $n_{i,r-1}$. Thus, at most half of the total number of samples can be used by any optimal local search algorithm.

The randomization in step (a) that precludes sampling multiple instances with the same sample size is motivated by the following example. Assume that A_1 converges to the correct estimate, while all the other algorithms $A_2, \ldots, A_K$ produce the same estimate in each round, independently of their samples, that is inferior to the estimates of A_1. If we use the randomization, half of the calls to compute f will be made by A_1, but without the randomization this would drop down to $1/K$ as in each round we would sample each algorithm. The randomization is used to exclude such pathological cases.

The following proposition shows that the algorithm is consistent.

Proposition 1. *Assume that $g_\delta(n)$ is positive and monotone decreases to 0 as $n \to \infty$. Then the* METAMAX(K) *algorithm is consistent in the sense that $\hat{f}_r \leq f(x^*)$ for all r, and*

$$f(x^*) - \lim_{r\to\infty} \hat{f}_r = \min_{i=1,\ldots,K} \left\{ f(x^*) - \lim_{n\to\infty} \hat{f}_{i,n} \right\}.$$

Proof. The proof follows trivially from the fact that each algorithm is selected infinitely often, that is, $\lim_{r\to\infty} n_{i,r} = \infty$. To see the latter, we show that in

every K rounds the number of samples received by the least used algorithm, that is, $\min_{i=1,\dots,K} n_{i,r}$, is guaranteed to increase by one. That is, for all $k \geq 0$,

$$\min_{i=1,\dots,K} n_{i,kK} \geq k. \tag{4}$$

As described above, in each round we select exactly one of the algorithms with minimal sample size. Thus, if there are k such algorithms, the minimal sample size will increase in k steps, which completes the proof. □

Let $\hat{f}_i^* = \lim_{r\to\infty} \hat{f}_{i,n_{i,r}}$ be the asymptotic estimate of algorithm A_i for $f(x^*)$, and let $\hat{f}^* = \max_{1\leq i\leq K} \hat{f}_i^*$ denote the best estimate achievable using algorithms $A_1, \dots, A_K$. Let $O \subset \{1, \dots, K\}$ be the set of optimal algorithms that converge to the best estimate $\hat{f}^*$ (for these algorithms), and let $|O|$ denote the cardinality of O (i.e., the number of optimal algorithm instances). The next lemma shows that if $i \notin O$, then A_i is not sampled at a round r if it has been sampled too often so far.

Lemma 1. *Suppose Assumption 1 holds, and let*

$$\Delta = \hat{f}^* - \max_{i\notin O} \hat{f}_i^*$$

denote the margin between the estimates of the best and the second best algorithms. Then there is an $R^{(\delta)} > 0$ such that A_i is not sampled by MetaMax(K) *at a round $r + 1 > R^{(\delta)}$ with probability at least $1 - K\delta$ given the starting points $X_{i,1}, i = 1, \dots, K$, if*

$$n_{i,r} > g_\delta^{-1}\left(g_\delta(\min_j n_{j,r})\left(1 - \frac{\hat{f}_i^* + \Delta/2}{\hat{f}^*}\right)\right) \tag{5}$$

simultaneously for all $i \notin O$.

Proof. For each $i = 1, \dots, K$, the conditions of Assumption 1 and (4) imply that there exists an $R_i^{(\delta)} > 0$ such that for all $r > R_i^{(\delta)}$

$$\mathbb{P}\left(\hat{f}_i^* - \hat{f}_{i,n_{i,r}} \leq \Delta/2 \;\middle|\; X_{i,1}\right) \geq 1 - \delta.$$

Note that the above inequality holds simultaneously for all $r > R_i^{(\delta)}$ since the difference $\hat{f}_i^* - \hat{f}_{i,n_{i,r}}$ is a non-increasing sequence in r. By the union bound, for any $r > R^{(\delta)} = \max\{R_1^{(\delta)}, \dots, R_K^{(\delta)}\}$

$$\mathbb{P}\left(\hat{f}_i^* - \hat{f}_{i,n_{i,r}} \leq \Delta/2 \text{ for all } i = 1, \dots, K | X_{1,1}, \dots, X_{K,1}\right) \geq 1 - K\delta. \tag{6}$$

This implies that after $R^{(\delta)}$ rounds the algorithm will pick, with high probability, the estimate of one of the best algorithms. It is easy to see that, given the starting points $X_{i,1}, i = 1, \dots, K$, if (6) holds then an algorithm A_i with $i \notin O$ cannot be

sampled at round $r+1$ if $n_{i,r} \geq n_{I_r,r}$ (that is, if it has been sampled more than an algorithm with a better maximum estimate). Furthermore, if $n_{i,r} < n_{I_r,r}$, it is easy to see that A_i is not sampled at a round $r+1$ if

$$\frac{\hat{f}_r - \hat{f}_{i,r}}{g_\delta(n_{i,r}) - g_\delta(n_{I_r,r})} > \frac{\hat{f}_r - 0}{g_\delta(\min_j n_{j,r}) - g_\delta(n_{I_r,r})}$$

since the line connecting $(g_\delta(\min_j n_{j,r}), 0)$ and $(g_\delta(n_{I_r,r}), \hat{f}_{I_r,n_{I_r,r}})$ is a lower bound to the upper convex hull of $\mathcal{P}_{r+1}$ (defined by (3)). Since $\hat{f}_r - \hat{f}_{i,r} \geq \hat{f}^* - \hat{f}_i^* - \Delta/2$ and $\hat{f}^* \geq \hat{f}_r$, given the starting points $X_{i,1}$, A_i is not selected at round $r+1$ simultaneously for all i, with probability at least $1 - K\delta$, if

$$\frac{\hat{f}^* - \hat{f}_i^* - \Delta/2}{g_\delta(n_{i,r}) - g_\delta(n_{I_r,r})} > \frac{\hat{f}^*}{g_\delta(\min_j n_{j,r}) - g_\delta(n_{I_r,r})}. \tag{7}$$

The latter is equivalent to the upper bound

$$g_\delta(n_{i,r}) < g_\delta(\min_j n_{j,r}) \left(1 - \frac{\hat{f}_i^* + \Delta/2}{\hat{f}^*}\right) + g_\delta(n_{I_r,r}) \frac{\hat{f}_i^* + \Delta/2}{\hat{f}^*}.$$

Taking into account that g_δ is non-increasing and positive, this is surely satisfied if (5) holds. □

A simple corollary of the above lemma is that if the local search algorithm converges fast enough (exponentially with a problem dependent rate, or faster than exponential) then half of the samples taken from f correspond to optimal algorithm instances.

Corollary 1. *Assume that the performance of the algorithms $A_i, i = 1, \ldots, K$ are not all the same, that is, $|O| < K$, and that the conditions of Assumption 1 hold. Furthermore, suppose that*

$$\limsup_{n\to\infty} \frac{g_\delta(n+1)}{g_\delta(n)} < \min_{i \notin O} \left\{ 1 - \frac{\hat{f}_i^* + \Delta/2}{\hat{f}^*} \right\}. \tag{8}$$

Then asymptotically at least half of the sampling of f in MetaMax(K) *corresponds to an optimal algorithm. That is,*

$$\mathbb{P}\left(\liminf_{r\to\infty} \frac{\sum_{i\in O} n_{i,r}}{\sum_{i=1}^K n_{i,r}} \geq \frac{1}{2} \;\middle|\; X_{1,1}, \ldots, X_{K,1} \right) \geq 1 - K\delta.$$

Proof. We show that A_i is not chosen for large enough r if $n_{i,r} > \min_j n_{j,r}$. By Lemma 1, it is sufficient to prove that, for large enough r,

$$\min_j n_{j,r} + 1 > g_\delta^{-1}\left(g_\delta(\min_j n_{j,r}) \left(1 - \frac{\hat{f}_i^* + \Delta/2}{\hat{f}^*}\right)\right).$$

The latter is clearly satisfied by (8) as $\lim_{r\to\infty} \min_j n_{j,r} = \infty$ by (4). This fact finishes the proof. □

MetaMax: A multi-start strategy with infinitely many algorithm instances.

Parameters: $0 < \delta < 1$, $g_\delta(n)$.
Initialization: Set $K = 0$. For each round $r = 1, 2, \ldots$

(a) Increase the value of K by 1. Initialize algorithm A_K by setting $n_{K,r-1} = 0, \hat{f}_{K,0} = 0$.
(b) For $i = 1, \ldots, K$ select algorithm A_i if there exists a $\hat{c} > 0$ such that

$$f_{i,n_{i,r-1}} + \hat{c} g_\delta(n_{i,r-1}) > f_{j,n_{j,r-1}} + \hat{c} g_\delta(n_{j,r-1}) \tag{9}$$

for all $j \neq i$ such that $n_{i,r-1} \neq n_{j,r-1}$. If there are several values of i selected that have the same sample number $n_{i,r-1}$ then keep only one of these selected uniformly at random.
(c) Step each selected A_i, and update variables. That is, set $n_{i,r} = n_{i,r-1} + 1$ if A_i is selected, and $n_{i,r} = n_{i,r-1}$ otherwise. For each selected A_i sample $f(X_{i,n_{i,r}})$ and compute the new estimates $\widehat{X}_{i,n_{i,r}}$ and $\hat{f}_{i,n_{i,r}}$.
(d) Let $I_r = \operatorname{argmax}_{i=1,\ldots,K} \hat{f}_{i,n_{i,r}}$ denote the index of the algorithm with the currently largest estimate of $f(x^*)$, and estimate the location of the maximum with $\widehat{X}_r = \widehat{X}_{I_r,n_{I_r,r}}$ and its value with $\hat{f}_r = \hat{f}_{I_r,n_{I_r,r}}$.

Fig. 2. The MetaMax algorithm

The above corollary immediately yields the following result on the rate of convergence of the MetaMax(K) algorithm.

Corollary 2. *Assume the conditions of Corollary 1 hold. Then for any $\delta > 0$ and $\epsilon > 0$ there exists a threshold $T_{\delta,\epsilon} > 0$ depending on the starting points $X_{1,1}, \ldots, X_{1,K}$ such that if the* MetaMax(K) *algorithm is run for r rounds such that the total number of samples $T = \sum_{i=1}^{K} n_{i,r}$ satisfies $T > T_{\delta,\epsilon}$, then, with probability at least $(1 - (K + |O|)\delta)$,*

$$\hat{f}^* - \hat{f}_r \leq c g_\delta \left(\frac{T}{(2+\epsilon)|O|} \right).$$

3.2 Unbounded Number of Instances

It is clear that if the local search algorithms are not consistent, then, despite its favorable properties, the MetaMax(K) strategy is inconsistent, too. However, if we increase the number of algorithms in each round, then we get the consistency from random search, while still keeping the reasonably fast convergence rate from MetaMax(K). This results in the multi-start strategy MetaMax, shown in Figure 2.

Since the MetaMax strategy includes a random search (the number of algorithms tends to infinity as the length of each round is finite), the algorithm is consistent:

Proposition 2. *The strategy* MetaMax *is consistent. That is,*

$$\lim_{r\to\infty} \hat{f}_r = f(x^*).$$

Under Assumption 1 we can prove, similarly to Lemma 1, that any suboptimal algorithm is selected only finitely many times.

Lemma 2. *Suppose Assumptions 1 holds, and assume that an instance A_i is suboptimal, that is, $\hat{f}_i^* = \lim_{r\to\infty} \hat{f}_{i,n_{i,r}} < f(x^*)$. Let*

$$\Delta_i = f(x^*) - \hat{f}_i^*.$$

Then, for sufficiently large r, A_i is not sampled at a round $r+1$, with probability at least $1-2\delta$, if

$$n_{i,r} > \max\left\{ g_\delta^{-1}\left(g_\delta(0)\left(1 - \frac{\hat{f}_i^* + \Delta_i/2}{f(x^*)}\right)\right),\ g_\delta^{-1}\left(\frac{\Delta_i}{2c}\right)\right\}.$$

Proof. By the assumptions of the lemma there exists a positive integer $R > 0$ such that $f(x^*) - \hat{f}_R < \Delta_i/2$ with probability at least $1-\delta$. (A crude estimate of R can be obtained by considering the probabilities that $X_{r,1}$ falls in a sufficiently small neighborhood of x^*. Better values of R can be calculated by considering the probability of choosing a starting point such that the corresponding algorithm converges to a global optimum, and then considering the rate of convergence of this local search algorithm.) By Assumption 1, $\hat{f}_i^* - \hat{f}_{i,n} < \Delta_i/2$ with probability at least $1-\delta$ for all $n > n_i = g_\delta^{-1}\left(\frac{\Delta_i}{2c}\right)$. Thus, by the union bound, for any round $r+1 > R$ such that $n_{i,r} > g_\delta^{-1}\left(\frac{\Delta_i}{2c}\right)$, we have, with probability at least $1-2\delta$, simultaneously

$$f(x^*) - \hat{f}_r < \Delta_i/2 \qquad \text{and} \qquad \hat{f}_i^* - \hat{f}_{i,n_{i,r}} < \Delta_i/2. \tag{10}$$

Now a similar argument as in Lemma 1 finishes the proof. Note that compared to (5) we replaced $\min_i n_{i,r}$ with 0 since a new algorithm is started in every round with $n_{r+1,r} = 0$. □

Experimental results in Section 4 show that K, the number of algorithm instances, increases sublinearly: in all simulations K was of the order $T/\ln(T)$ with $K < 2T/\ln T$. It is interesting to note that this is roughly the same as the number of algorithm instances introduced by the algorithm Luby [17] (see Section 4 for more details). If Assumption 1 is satisfied with $\delta = 0$ and there are only finitely many local maxima of f, this fact gives rise to the following heuristic derivation for the convergence rate of MetaMax. Let p be the probability that a random starting point is chosen such that the resulting algorithm converges to a global optimum. Then, for T large enough, there are approximately

$(1-p)T/\ln T$ suboptimal algorithms, and, after some initial phase with S samples, each of them can be sampled at most N times for some integer N. Then the $pT/\ln T$ optimal algorithms are sampled at least $T - N(1-p)T/\ln T - S$ times, so one of them is sampled at least $(T-N(1-p)T/\ln T-S)/(pT/\ln T) = O(\ln T)$ times, resulting in a convergence rate of $cg_0(\kappa \ln T)$ for some $\kappa > 0$.

4 Experiments

In the experiments we compared the two versions of the MetaMax algorithm with four algorithms including two with fixed (K), and two with variable number of algorithm instances (arms). We used the SPSA ([18], Simultaneous Perturbation Stochastic Approximation) as the base local search algorithm in all cases. SPSA is a local search algorithm with a sampling function that uses gradient descent with a stochastic approximation of the derivative: at the actual location $X_t = (X_{t,1}, \ldots, X_{t,d})$, SPSA estimates the derivative of f by

$$\tilde{f}_{t,l}(X_{t,l}) = \frac{f(X_t + \phi_t B_t) - f(X_t - \phi_t B_t)}{2\phi_t B_{t,l}},$$

where the $B_{t,l}$ are i.i.d. Bernoulli random variables that are the components of the vector B_t, and then uses the sampling function $s_t(X_t) = X_t + a_t \tilde{f}_t(X_t)$ to choose the next point to be sampled, that is,

$$X_{t+1,l} = X_{t,l} + a_t \tilde{f}_{t,l}(X_{t,l})$$

for $l = 1, \ldots, d$.

In the implementation of the algorithm we have followed the guidelines provided in [19], with the gain sequence $a_t = a/(A + t + 1)^\alpha$, and perturbation size $\phi_t = \phi/(t+1)^\gamma$, where $A = 60$, $\alpha = 0.602$ and $\gamma = 0.101$. The values of a and ϕ vary in the different experiments. Next to the two evaluations required at the perturbed points, we also evaluate the function at the current point X_t. The starting point is chosen randomly, and the function is evaluated first at this point.

The four reference algorithms the MetaMax(K) and MetaMax algorithms are compared to are the following:

Unif: This algorithm selects from a constant number of instances of SPSA uniformly. In our implementation the instance $I_t = t \bmod K$ is selected at time t, where K denotes the number of instances.

ThrAsc: The ThresholdAscent algorithm of [21]. The algorithm begins with selecting each of a fixed number of instances once. After this phase at each time step t ThrAsc selects the best s estimates produced so far by all algorithm instances $A_i, i = 1, \ldots, K$ in all the previous time steps, and for each A_i it counts how many of these estimates were produced by A_i. Denoting the latter value by $S_{i,t}$, at time t the algorithm selects the instance with index

$I_t = \text{argmax}_i U(S_{i,t}/n_{i,t}, n_{i,t})$, where $n_{i,t}$ is the number of times the ith instance has been selected up to time t,

$$U(\mu, n) = \mu + \frac{\alpha + \sqrt{2n\mu\alpha + \alpha^2}}{n}$$

and $\alpha = \ln(2TK/\delta)$. s and δ are the parameters of the algorithm, and in the experiments the best value for s appeared to be 100, while δ was set to 0.01.

RAND: The random search algorithm. It can be seen as running a sequence of SPSA algorithms such that each instance is used for exactly one step, which is the evaluation of the random starting point of the SPSA algorithm.

LUBY: The algorithm based on [17]. This method runs several instances of SPSA sequentially after each other, where the ith instance is run for t_i steps, with t_i defined by

$$t_i = \begin{cases} 2^{k-1}, & \text{if } i = 2^k - 1 \\ t_{i-2^{k-1}+1}, & \text{if } 2^{k-1} \le i < 2^k - 1 \end{cases}$$

The above definition produces a scheduling such that from the first $2^k - 1$ algorithm instances one is run for 2^{k-1} steps, two for 2^{k-2} steps, four for 2^{k-3} steps, and so on.

Both versions of the METAMAX algorithm were tested. Motivated by the fact that SPSA is known to converge to a global optimum exponentially fast if f satisfies some restrictive conditions [12], we chose a $g_\delta(n)$ that decays exponentially fast (recall that this is the rate of convergence to a local minimum). Furthermore, to control the exploration of the so far suboptimal algorithm instances heuristically, we allowed $g_\delta(n)$ to be a time-varying function, that is, it changes with t, the total number of samples seen so far. Thus, at round r we used $g_\delta(n) = e^{-n/\sqrt{t_r}}$, where $t_r = \sum_i n_{i,r-1}$.

For the algorithms with a fixed number of arms (METAMAX(K), UNIF, and THRASC), the number of arms K was set to 100 in the simulations, which turned out to be a good choice overall.

The multi-start algorithms were tested using two versions of a synthetic function, and by tuning the parameters of a learning algorithm on two standard data sets.

The synthetic function was a slightly modified[2] version of the Griewank function [13]:

$$f(x) = \sum_{l=1}^{d} \frac{4\pi^2 x_l^2}{100} - \prod_{l=1}^{d} \cos \frac{2\pi x_l}{\sqrt{l}} + 1$$

where $x = (x_1, \ldots, x_l)$ and the x_i were constrained to the interval $[-1, 1]$. We show the results for the two-dimensional and the ten-dimensional cases.

The parameters of SPSA were $a = 0.05$ and $\phi = 0.1$ for the two-dimensional case, and $a = 0.5$ and $\phi = 0.1$ for the ten-dimensional case. The performance

[2] The modification was made in order to have more significant differences between the values of the function at the global maximum and at other local maxima.

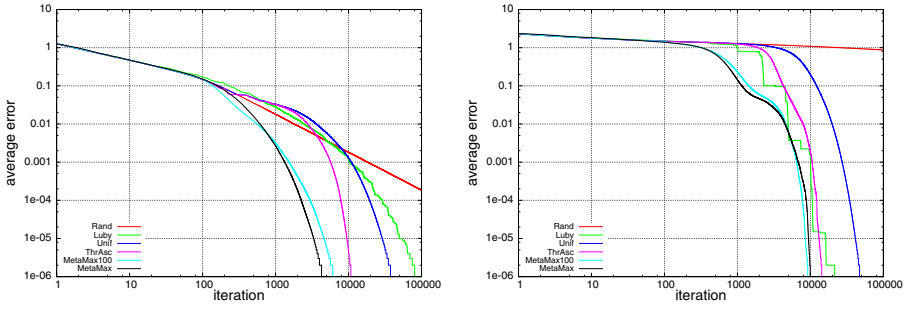

Fig. 3. The average error for the multi-start strategies on the two-dimensional (left) and ten-dimensional (right) modified Griewank function

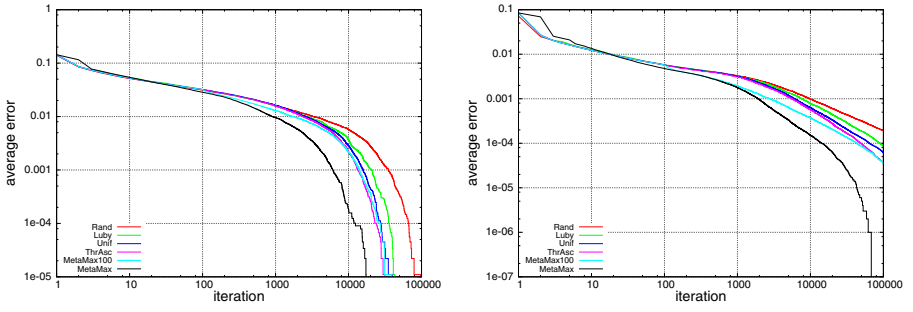

Fig. 4. The average error for the multi-start strategies on tuning the parameters of MultilayerPerceptron for the *vehicle* data set (left) and the *letter* data set (right)

of the search algorithms were measured by the error defined as the difference between the maximum value of the function (in this case 1) and the best result obtained by the search algorithm in a given number of steps. The results for the above multi-start strategies for the two- and the ten-dimensional test functions are shown in Figure 3. Each error curve is averaged over 10,000 runs, and each strategy was run for 100,000 steps (or iterations). One may observe that in both cases the two versions of the MetaMax algorithm converge the fastest. ThrAsc is better then Unif, while Luby seems fairly competitive with these two. The random search seems an option only for the low-dimensional function. Similar results were obtained for dimensions between 2 and 10.

For tuning the parameters of a learning algorithm, we have used two standard data sets from the UCI Machine Learning Repository [2]: *vehicle* and *letter*, and the MultilayerPerceptron learning algorithm of Weka [23]. Two parameters were tuned: the learning rate and the momentum, both in the range of $[0, 1]$. The size of the hidden layer for the MultilayerPerceptron was set to 8, while the number of epochs to 100. The parameters of the SPSA algorithm were $a = 0.5$ and $\phi = 0.1$. The rate of correctly classified items on the test set for *vehicle* using MultilayerPerceptron with varying values of the two parameters is shown

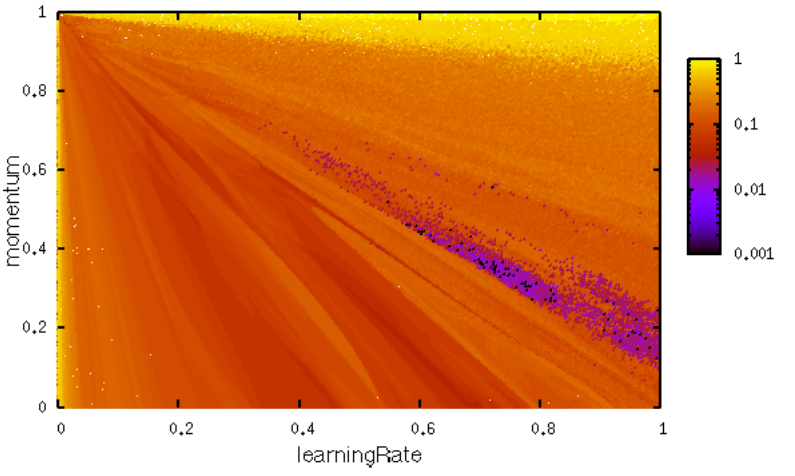

Fig. 5. Classification rate on the *vehicle* data set. The rates are plotted by subtracting them from 0.911112 and thus global optima are at the scattered black spots corresponding to a value equal to 0.001.

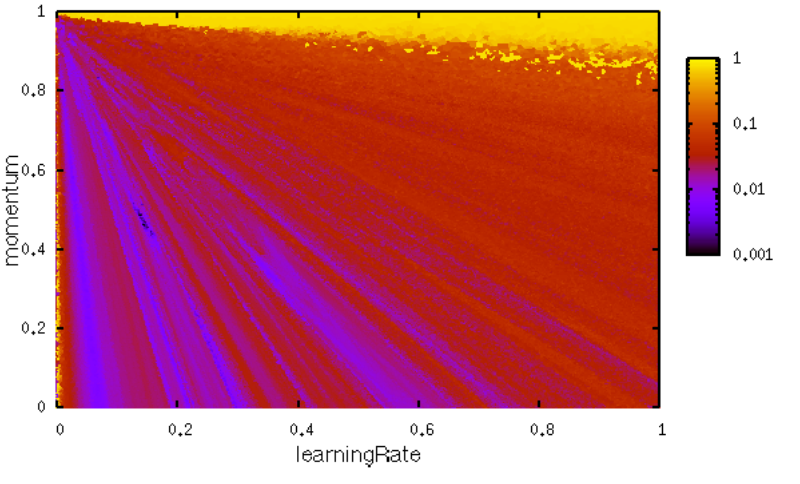

Fig. 6. Classification rate on the *letter* data set. The rates are plotted by subtracting them from .7515 and thus global optima are at the scattered black spots corresponding to a value equal to 0.001.

in Figure 5, with the highest rate being 0.910112. Similarly, the classification rate for *letter* is shown in Figure 6, with the highest rate being 0.7505.

The error rates of the optimized MultilayerPerceptron on the data sets *vehicle* and *letter* are shown in Figure 4, when the parameters of the learning algorithm were tuned by the multi-start strategies above. The error in these cases is the difference between the best classification rate that can be obtained (0.910112 and

0.7505, respectively) and the best classification rate obtained by the multi-start strategies in a given number of steps. The results shown are averages over 1,000 runs. We observe that the MetaMax algorithm (with an increasing number of arms) converged the fastest in average, the three strategies with a fixed number of arms had nearly identical results, Luby was slightly worse than these, and the random search was the slowest, although it did not perform as bad as for the synthetic functions.

In summary, the MetaMax algorithm (with an increasing number of arms) provided far the best performance in all tests, usually requiring an order of magnitude less steps to find the optimum than the other algorithms. E.g., for the *letter* data set only the MetaMax algorithm found the global optimum in all runs in 100,000 time steps. We can conclude that MetaMax converged faster than the other multi-start strategies investigated in all four test cases, with a notable advantage on the difficult surfaces induced by the classification tasks.

5 Conclusions

We provided a multi-start strategy for local search algorithms. The method assumes a convergence rate of the local search algorithm up to an unknown constant, and in every phase it gives preference to those local search algorithm instances that could converge to the best value for a particular range of the constant. Two versions of the algorithm were presented, one that is able to follow the performance of the best of a fixed number of local search algorithm instances, and one that, with gradually increasing the number of the local search algorithms, achieves global consistency. Some theoretical properties of the algorithms were explored, although the methods used do not seem to fully explain the superior behavior of the strategy in experiments, requiring further study of the problem.

References

1. Adam, K.: Learning while searching for the best alternative. Journal of Economic Theory 101, 252–280 (2001)
2. Asuncion, A., Newman, D.J.: UCI machine learning repository (2007)
3. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite time analysis of the multiarmed bandit problem. Machine Learning 47(2-3), 235–256 (2002)
4. Beck, C.J., Freuder, E.C.: Simple rules for low-knowledge algorithm selection. In: Régin, J.-C., Rueher, M. (eds.) CPAIOR 2004. LNCS, vol. 3011, pp. 50–64. Springer, Heidelberg (2004)
5. Carchrae, T., Beck, J.C.: Low-knowledge algorithm control. In: Proceedings of the 19th National Conference on Artificial Intelligence (AAAI), pp. 49–54 (2004)
6. Cicirello, V.A.: The max k-armed bandit: A new model of exploration applied to search heuristic selection. In: Proceedings of the Twentieth National Conference on Artificial Intelligence, pp. 1355–1361 (2005)
7. Cicirello, V.A., Smith, S.F.: Heuristic selection for stochastic search optimization: Modeling solution quality by extreme value theory. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 197–211. Springer, Heidelberg (2004)

8. Finkel, D.E., Kelley, C.T.: Convergence analysis of the direct algorithm. Technical Report CRSC-TR04-28, NCSU Mathematics Department (2004)
9. Gagliolo, M., Schmidhuber, J.: Learning dynamic algorithm portfolios. Annals of Mathematics and Artificial Intelligence 47(3–4), 295–328 (2006); AI&MATH 2006 Special Issue
10. Gagliolo, M., Schmidhuber, J.: Learning restart strategies. In: Veloso, M.M. (ed.) IJCAI 2007 — Twentieth International Joint Conference on Artificial Intelligence, vol. 1, pp. 792–797. AAAI Press, Menlo Park (2007)
11. Gagliolo, M., Schmidhuber, J.: Algorithm selection as a bandit problem with unbounded losses. Technical Report IDSIA - 07 - 08, IDSIA (July 2008)
12. Gerencsér, L., Vágó, Z.: The mathematics of noise-free SPSA. In: Proceedings of the IEEE Conference on Decision and Control, pp. 4400–4405 (2001)
13. Griewank, A.O.: Generalized descent for global optimization. Journal of Optimization Theory and Applications 34, 11–39 (1981)
14. Horn, M.: Optimal algorithms for global optimization in case of unknown Lipschitz constant. Journal of Complexity 22(1), 50–70 (2006)
15. Jones, D.R., Perttunen, C.D., Stuckman, B.E.: Lipschitzian optimization without the lipschitz constant. Journal of Optimization Theory and Applications 79(1), 157–181 (1993)
16. Kautz, H., Horvitz, E., Ruan, Y., Gomes, C., Selman, B.: Dynamic restart policies. In: Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI), pp. 674–681 (2002)
17. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of Las Vegas algorithms. Information Processing Letters 47, 173–180 (1993)
18. Spall, J.C.: Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. IEEE Transactions on Automatic Control 37, 332–341 (1992)
19. Spall, J.C.: Implementation of the simultaneous perturbation algorithm for stochastic optimization. IEEE Transactions on Aerospace Electronic Systems 34, 817–823 (1998)
20. Streeter, M.J., Smith, S.F.: An asymptotically optimal algorithm for the max k-armed bandit problem. In: Proceedings, The 21st National Conference on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conference, pp. 135–142 (2006)
21. Streeter, M.J., Smith, S.F.: A simple distribution-free approach to the max k-armed bandit problem. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 560–574. Springer, Heidelberg (2006)
22. Vilalta, R., Drissi, Y.: A perspective view and survey of meta-learning. Artificial Intelligence Review 18(2), 77–95 (2002)
23. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)