A MEMORY-BASED RASH OPTIMIZER

Mauro Brunato, Roberto Battiti, Srinivas Pasupuleti

April 26, 2006

Technical Report # DIT-06-023

A Memory-Based RASH Optimizer

Mauro Brunato, Roberto Battiti
Srinivas Pasupuleti
Dept. of Computer Science and Telecommunications
University of Trento, Italy,
brunato@dit.unitn.it

April 26, 2006

Abstract

This paper presents a *memory-based* Reactive Affine Shaker (M-RASH) algorithm for global optimization. The Reactive Affine Shaker is an adaptive search algorithm based only on the function values. M-RASH is an extension of RASH in which good starting points to RASH are suggested online by using Bayesian Locally Weighted Regression (B-LWR). Both techniques use the memory about the previous history of the search to guide the future exploration but in very different ways. RASH compiles the previous experience into a local search area where sample points are drawn, while locally-weighted regression saves the entire previous history to be mined extensively when an additional sample point is generated. Because of the high computational cost related to the B-LWR model, it is applied only to evaluate the potential of an initial point for a local search run. The experimental results, focussed onto the case when the dominant computational cost is the evaluation of the target f function, show that M-RASH is indeed capable of leading to good results for a smaller number of function evaluations.

1 Introduction

Like furniture is in the searching look of a carpenter walking in a forest, technology is in the eyes of the computer scientist both as an end (e.g., solving optimization and planning problems) and as a means by which larger and larger instances can be solved. It is now a truism that the growing availability of massive amounts of memory, starting from the eighties, opened new windows of opportunity for memory-based optimization techniques, in particular memory-based heuristics. The underlying assumption of a rich internal structure of most relevant optimization tasks makes techniques capable of *gradually learning* that structure potentially more powerful and effective than memory-less techniques. Notable examples are the use of *pattern databases* originally proposed by [6] to solve tile puzzles. In these problems, the final state is known and the sequence of moves to reach it, is to be determined. The database is used to obtain a lower bound on the cost to reach the goal from a given state in the search space, by looking up all possible subgoals, see also [10] for more accurate admissible heuristic evaluation functions.

A different context is that of stochastic local search [8], where one aims at minimizing a function f of discrete or continuous variables. In this case the optimal configuration is not known at the beginning and generating a trajectory by local search in the configuration space is a way to explore promising configurations aiming at discovering good local optima. The authors of this paper used history-sensitive (or memory-based) techniques to solve combinatorial problems [3] and continuous optimization problems. The recent publication [2] summarizes the methods and the main applications, while [1] is dedicated to data structures based on hashing and dynamic sets to support history-sensitive heuristics.

In this small contribution, because of the limited space and because we think it could offer a different point of view we concentrate on a recent exploration related to the usage of models based on memory in order to speed up a simple stochastic search method denoted as Reactive Affine Shaker proposed in [4]. In the following sections the basic building blocks, the RASH local search heuristic and the B-LWR modeling technique, are described. Next, a combination of the two techniques is proposed. Finally, experimental results on classical optimization problems are discussed.

2 Building blocks

In the following discussion, we make the assumption that the dominant computational cost is related to evaluating the target function f at trial points. This assumption is justified in many practical applications, for example when the evaluation of f requires running a lengthy simulation, or even running an industrial plant and measuring the output. It is in these cases that the use of memory is worth the effort and to make the assumption explicit we will discuss about results obtainable as a function of the number of f evaluations in the next part of the paper. A more detailed analysis taking into account the trade-off between the overhead involved in the usage of memory for cases when this is not negligible is in preparation and not shown in this paper because of space reasons.

The proposed memory-based technique, M-RASH, is based on two major components: an efficient local search heuristic, RASH, for rapidly finding local minima, and a statistically sound method, Bayesian Locally-weighted Regression, to model and predict its global behavior. In this Section we briefly describe these two components.

2.1 The RASH heuristic

The Reactive Affine Shaker Heuristic [4], RASH for short, is a self-tuning local search algorithm based on the framework proposed in [13], where no prior knowledge is assumed on the function to be minimized and only evaluations at arbitrary values of the independent variables are allowed. The RASH heuristic tries to rapidly move towards better objective values by maintaining and updating a small "search region" $\mathcal{R}$ around the current point $\boldsymbol{x}$.

f	Function to minimize
$\boldsymbol{x}$	Initial point
$\mathcal{R}$	Search region
$\boldsymbol{\Delta}$	Current displacement

```
1.  function RASH ($f$, $\boldsymbol{x}$)
2.      $\mathcal{R} \leftarrow$ small istropic set around $\boldsymbol{x}$
3.      while (local termination condition is not met)
4.          Pick $\boldsymbol{\Delta} \in \mathbb{R}^d$ such that $\boldsymbol{x} + \boldsymbol{\Delta}, \boldsymbol{x} - \boldsymbol{\Delta} \in \mathcal{R}$
5.          if $f(\boldsymbol{x} + \boldsymbol{\Delta}) < f(\boldsymbol{x})$
6.              $\boldsymbol{x} \leftarrow \boldsymbol{x}$ + $\boldsymbol{\Delta}$;
7.              Extend $\mathcal{R}$ along $\boldsymbol{\Delta}$
8.              Center $\mathcal{R}$ on $\boldsymbol{x}$
9.          else if $f(\boldsymbol{x} - \boldsymbol{\Delta}) < f(\boldsymbol{x})$
10.             $\boldsymbol{x} \leftarrow \boldsymbol{x}$ - $\boldsymbol{\Delta}$;
11.             Extend $\mathcal{R}$ along $\boldsymbol{\Delta}$
12.             Center $\mathcal{R}$ on $\boldsymbol{x}$
13.         else
14.             Reduce $\mathcal{R}$ along $\boldsymbol{\Delta}$
15.     return $\boldsymbol{x}$;
```

Figure 1: The RASH algorithm

The use of memory in RASH is limited: the entire previous history of the search (the trajectory of the generated sample points and the outcome of the evaluations) is summarized through a *dynamic search region*, intended to zoom in onto the promising areas where to find points better than the current best.

The efficiency of RASH lies in the ability to reshape the search region $\mathcal{R}$ according to the occurrence or lack of success during the last step: if a step in a certain direction yields a better objective value, then $\mathcal{R}$ is expanded along that direction; it is reduced otherwise. Therefore, once a promising direction is found, the probability that subsequent steps will follow the same direction is increased, and search shall proceed more and more aggressively in that direction until bad results reduce its prevalence. The algorithm is outlined in Fig. 1.

The algorithm starts with an isotropic search region centered around the initial point (line 2). Next, new trail points are repeatedly generated (line 4). If the resulting point $\boldsymbol{x} + \boldsymbol{\Delta}$ yields a lower objective value (line 5 and following), then the current position is updated and $\mathcal{R}$ is expanded along the direction of $\boldsymbol{\Delta}$. To increase the probability of finding a better point, if $\boldsymbol{x} + \boldsymbol{\Delta}$ does not lead to an improvement, also $\boldsymbol{x} - \boldsymbol{\Delta}$ is checked (line 9 and following). If none of the points improves the objective value, then the search region is reduced along the direction of $\boldsymbol{\Delta}$ (line 14) and the current position is not updated. This sequence of steps is repeated until a local termination condition is verified. Common criteria to terminate the search are the number of iterations, the size of the search region (if too small, it indicates that no precise direction for improvement

f	Function to minimize
$\boldsymbol{x}$, $\boldsymbol{x}'$	Initial and final position of run
bestPoint, *bestValue*	Best position found and its value

```
1. function RepeatedRASH ($f$)
2.     bestValue ← $+\infty$
3.     while (overall termination condition is not met)
4.         $\boldsymbol{x}$ ← random point in $f$ domain
5.         $\boldsymbol{x}$' ← RASH ($f$, $\boldsymbol{x}$)
6.         if $f(\boldsymbol{x}') <$ bestValue
7.             bestPoint ← $\boldsymbol{x}$
8.             bestValue ← $f(\boldsymbol{x}')$
9.     return bestPoint
```

Figure 2: The Repeated RASH algorithm

can be detected, therefore the system is already close to a local minimum), a large number of iterations without further improvement.

To keep an acceptable level of complexity, the search region is implemented as a box defined by d independent vectors ($\boldsymbol{b}_1 \ldots \boldsymbol{b}_d$), where d is the number of dimensions of the search domain. Shape modifications are implemented as affine transformations of these vectors, as described in the following equation:

$$\forall j \; \boldsymbol{b}_j \leftarrow \left(\boldsymbol{I} + (\rho - 1) \frac{\boldsymbol{\Delta}\boldsymbol{\Delta}^T}{\|\boldsymbol{\Delta}\|^2} \right) \boldsymbol{b}_j$$

The value of ρ is of 1.2 for expansions and 0.8 for compressions of the search region respectively. The easiest, although effective, way of improving the performance of the algorithm is to restart the search from a random point as soon as the local termination condition is verified, as shown in Fig. 2. This corresponds to having a population of searchers, each unaware of the others.

2.2 Bayesian Locally Weighted Regression

On the coordinate axis of "amount of memory usage", RASH stays at a very low level, while the extreme position is occupied by methods storing the entire history in memory aiming at mining it in the most flexible and effective way in order to generate a single additional trial point.

In particular, Bayesian Locally Weighted Regression [5, 11, 7], denoted as B-LWR, is characterized as a *lazy* memory-base technique where all points and evaluations are stored and a specific model is built *on-demand* for a specified query point. The usual power of Bayesian techniques derives from the explicit specification of the modeling assumptions and parameters (for example, a *prior distribution* can model our initial knowledge about the function) and the possibility to model not only the expected values but entire probability distributions, so that for example confidence intervals can be derived to quantify the confidence in the expected values.

B-LWR is the second fundamental building block considered to complement the M-RASH heuristic. B-LWR is a learning technique used to build a model out of data provided, for instance, by a stochastic or noisy function such as the outcome of an experiment.

The B-LWR algorithm relies on a set of n *sample data* $\{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)\}$ where $y_i \in \mathbb{R}$ is the outcome of a stochastic function evaluation on independent variable $\boldsymbol{x}_i \in \mathbb{R}^d$. To predict the outcome of an evaluation on a point $\boldsymbol{q}$ (named a *query point*), linear regression is applied to sample points. To enforce locality in the determination of regression parameters each sample point is assigned a weight that decreases with its distance from the query point. A common *kernel function* used to set the relationship between weight and distance is

$$w_i = e^{-\frac{\|\boldsymbol{x}_i - \boldsymbol{q}\|^2}{K}},$$

where K is a parameter measuring the kernel width, i.e. the sensitivity to far sample points.

The occasional lack of sample points near the query point would pose problems in estimating the regression coefficients with a simple linear model. Hence Bayesian regression is used, where we can specify prior information about what values the coefficients should have when there is not enough data to determine them. Bayesian LWR commonly assumes wide, weak Gaussian prior distribution of the coefficients of the regression model and a wide Gamma prior on the inverse of the noise covariance.

The linear regression model with Gaussian noise σ^2 is

$$y_i = \boldsymbol{x}_i^T \boldsymbol{\beta} + \epsilon,$$

where $\boldsymbol{\beta}$ is the vector of parameters of the linear model. Note that a constant 1 is appended to all input vectors $\mathbf{x_i}$ to include a constant term in the regression, so that the dimensionality of all equations is actually $d+1$. The samples can be collected in a matrix equation:

$$\boldsymbol{y} = X\boldsymbol{\beta}$$

where X is an $n \times (d+1)$ matrix whose ith row is $\boldsymbol{x}_i^T$ (complemented with a 1 entry to account for the constant term) and $\boldsymbol{y}$ is a vector whose ith element is y_i.

The task is to estimate the coefficients $\boldsymbol{\beta} = (\beta_0 \dots \beta_d)$. The prior assumption on $\boldsymbol{\beta}$ is that it is distributed according to a multivariate Gaussian of mean 0 and covariance matrix Σ, and the prior assumption on σ is that $1/\sigma^2$ has a Gamma distribution with k and θ as the shape and scale parameters. Since we use a weighted regression, each data point and the output response are weighted using Guassian weighting function. In matrix form, the weights for the data points are described in $n \times n$ diagonal matrix $W = \text{diag}(w_1, \dots, w_n)$. The prior assumes $\Sigma = \text{diag}(20^2, \dots, 20^2)$ for $\boldsymbol{\beta}$ distribution and $k = 0.8$, $\theta = 0.001$ for Gamma distribution.

The model local to the query point $\boldsymbol{q}$ is predicted by using the marginal posterior distribution of $\boldsymbol{\beta}$ whose mean is estimated as

$$\bar{\boldsymbol{\beta}} = (\Sigma^{-1} + X^T W^2 X)^{-1} (X^T W^2 \boldsymbol{y}). \tag{1}$$

The matrix $\Sigma^{-1} + X^T W^2 X$ is the weighted covariance matrix, supplemented by the effect of the $\boldsymbol{\beta}$ priors. Its inverse is denoted by V_β. The variance of the Gaussian noise based on n data points is estimated as

$$\sigma^2 = \frac{2\theta + (\boldsymbol{y}^T - \boldsymbol{\beta}^T X^T) W^2 \boldsymbol{y}}{2k + \sum_{i=1}^n w_i^2}.$$

The estimated covariance matrix of the $\boldsymbol{\beta}$ distribution is then calculated as

$$\sigma^2 V_\beta = \frac{(2\theta + (\boldsymbol{y}^T - \boldsymbol{\beta}^T X^T) W^2 \boldsymbol{y})(\Sigma^{-1} + X^T W^2 X)}{2k + \sum_{i=1}^n w_i^2}.$$

The degrees of freedom are given by $k + \sum_{i=1}^n w_i^2$. Thus the predicted output response for the query point $\mathbf{q}$ is

$$\hat{y}(\boldsymbol{q}) = \boldsymbol{q}^T \bar{\boldsymbol{\beta}},$$

while the variance of the mean predicted output is calculated as:

$$\text{Var}(\hat{y}(\boldsymbol{q})) = \boldsymbol{q}^T V_\beta \boldsymbol{q} \sigma^2. \qquad (2)$$

3 A global model for a local search heuristic

Locally Weighted Regression is an efficient way to model stochastic dependencies, such as those arising from experimental data. In this Section we define a local search heuristic as a stochastic function, and show how we use LWR (all the references to LWR mean Bayesian-LWR) to model its global behavior and predict the position of good starting points.

3.1 Local search algorithms as stochastic functions

Let f be a real-valued function defined over a limited domain $D \subset \mathbb{R}^d$. Let L be a local optimization heuristic, and let L_f the algorithm obtained by applying L to function f. L_f works by starting from an initial point $\boldsymbol{x}_1 \in D$ and generating a trajectory $(\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$, where N is the number of steps the algorithm performs before a termination condition is verified. If we treat the initial point $\boldsymbol{x}_1$ as an independent variable (i.e., not randomly generated by the algorithm itself, but fed as a parameter), the algorithm L_f can be seen as a function mapping the initial point of the trajectory to the smallest function value found along the trajectory:

$$\begin{aligned} L_f : D &\to \mathbb{R} \\ \boldsymbol{x}_1 &\mapsto \min_{i=1,\dots,N} f(\boldsymbol{x}_i). \end{aligned} \qquad (3)$$

Note that, since L is a stochastic heuristic relying on random choices, the trajectory is stochastic too, and L_f must be regarded as a stochastic function.

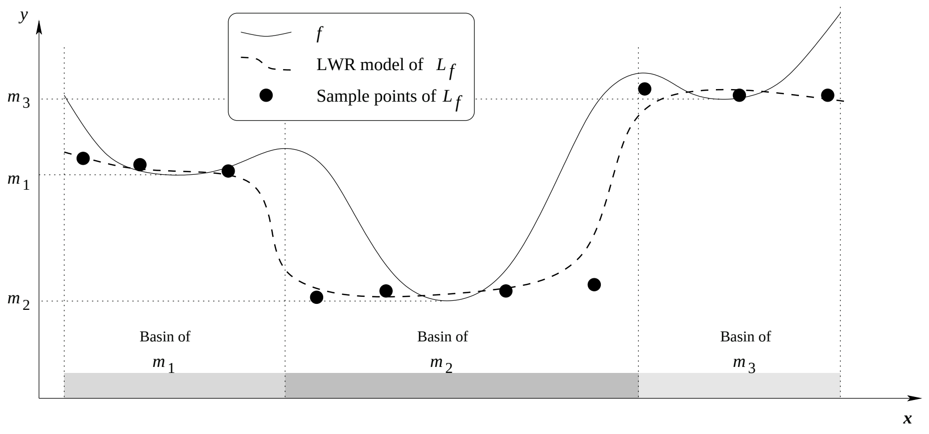

Figure 3: Modeling the local search algorithm L_f

3.2 An LWR model of the stochastic local search transformation

The stochastic function L_f models the transformation executed by local search, from an initial point to the local minimum point in a given attraction basin. After some runs of local search have been executed, one begins to derive knowledge about the structure of the search space, about which region is mapped to which local minima, and about a possible large-scale structure of the local minima showing the way to the most promising areas. Of course, the so-called “no free lunch” theorems of global optimization [14] imply that these techniques will not be effective for general functions (for sure they will not be effective if the value at one point is not related to values at nearby points), but most optimization problems of real interest are indeed characterized by a rich structure which can be profitably mined.

The integration proposed in this paper considers the LWR to model the transformation executed by L_f, therefore to evaluate the potential of future initial points to lead to promising local minima. For each run of the stochastic local search, the memory-based model will be mined to identify the next initial point. Other options are possible, like the consideration of an LWR model for describing the original function f. This second hypothesis is not considered here because of space reasons and because it leads to a more CPU-time consuming algorithm, but see [9] for an independent preliminary investigation.

To visualize the effect of the L_f transformation and the related modelling by LWR, Fig. 3 describes the application of a LWR technique to L_f in order to model it. Function f has three local minima, whose values are represented as m_1, m_2 and m_3, with m_2 as the global minimum value. Black dots represent sample points, of the form $(\boldsymbol{x}, L_f(\boldsymbol{x}))$, i.e., each is obtained by generating an initial value $\boldsymbol{x}$, feeding it to the local search algorithm, and retrieving the minimum value of f found along the subsequent trajectory. If the search algorithm makes local moves, as is the case with RASH, the sample points will approximately outline a stepwise function, constant in every attraction basin cor-

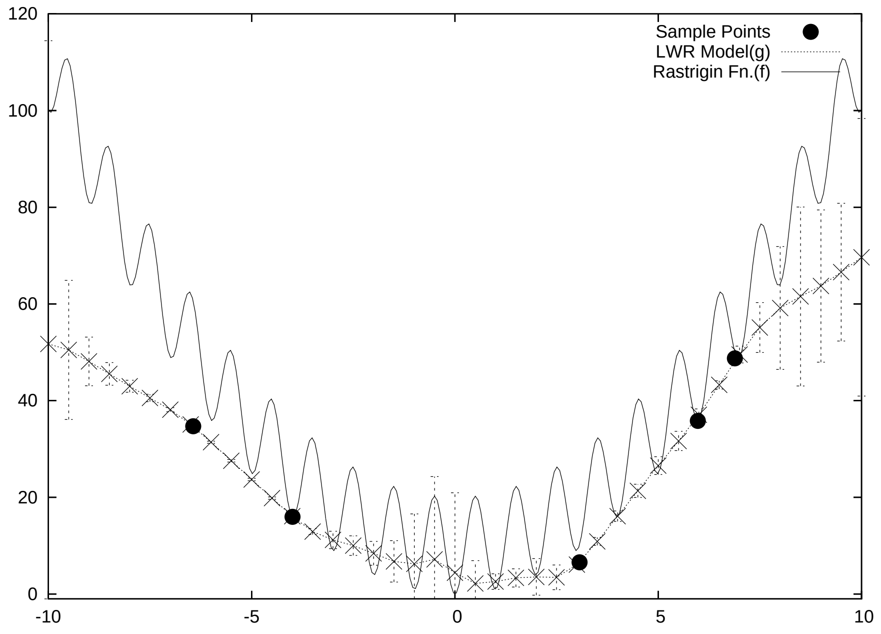

Figure 4: Modeling the Rastrigin function in 1 dimension

f	Function to minimize
D	Domain of f
g	B-LWR model of L_f, initially empty
n	Number of initial sample points in g

```
1.  function BLWR_RASH ($f$, $n$)
2.      for $i \leftarrow 1$ to $n$
3.          $\boldsymbol{x} \leftarrow$ random point in $D$
4.          $\boldsymbol{x}' \leftarrow$ RASH ($f$, $\boldsymbol{x}$)
5.          $g$.addSamplePoint ( $\boldsymbol{x}$, $f(\boldsymbol{x}')$)
6.      while (termination condition is not met)
7.          $\boldsymbol{x} \leftarrow$ Repeated_RASH ($g$)
8.          $\boldsymbol{x}' \leftarrow$ RASH ($f$, $\boldsymbol{x}$)
9.          $g$.addSamplePoint ( $\boldsymbol{x}$, $f(\boldsymbol{x}')$)
10.     return best point found
```

Figure 5: The memory-based M-RASH heuristic

responding to a given local minimum. Let's note that the smooth approximation to the stepwise function is actually useful to give the algorithm a direction to follow to reach promising areas, while an exact constant model on the plateau would not give such direction hint. The LWR model, shown in thick dashed line, is a smoothed out version of this stepwise function. Figure 4 shows a practical example executed on the 1-dimensional Rastrigin function. The sequence of sample points models the trend of local minima towards the global minimum, situated an $x = 0$. Note that the sample points represent the initial point and the the final function value as a result of applying the local search technique. The error bars indicate the variance on the predicted function value using the LWR model.

The LWR model of L_f (derived in Figs. 3 and 4) is in turn minimized in order to find the best suitable starting point for the subsequent run of L_f, as described in the following Section, where the technique just described is applied to the RASH heuristic.

3.3 The M-RASH Heuristic

Fig. 5 presents the pseudo-code for the M-RASH heuristic. The parameters are the function f to be minimized and the number of initial sample points in the model. Since we are using the Bayesian version of LWR with prior coefficient distribution, we are not forced to insert into the model a minimum number of points before it becomes useful.

The model g is initially empty; we assume that it can be evaluated at a query point as a real-valued function (in our C++ implementation, the B-LWR model implements a function interface), and that it can be updated by adding new points by calling the method g.`addSamplePoint` $(\boldsymbol{x}, y)$. The RASH local search algorithm is made available through the two function calls described in Fig. 1 and Fig. 2. In particular, it is important to remember that

Table 1: Benchmarks for simulations

Function Name	d	Mathematical Representation
Rosenbrock	10	$\sum_{i=1}^{d}\big(100(x_{i+1}-x_i^2)^2+(x_i-1)^2\big)$
Rastrigin	10	$\sum_{i=1}^{d}\big(x_i^2-10\cos 2\pi x_i+10\big)$
Schaffer	2	$0.5-\frac{(\sin\sqrt{x^2+y^2})^2-0.5}{(1.0+0.001(x^2+y^2))^2}$

- `RASH` $(f,\, \boldsymbol{x})$ is a single-run local search which starts at the initial point $\boldsymbol{x}$ and outputs the best point found over the function f until a termination condition is verified.
- `Repeated_RASH`(f) allows search to restart as soon as it detects that it is stuck at a local minimum. The search shall always start from a random point within D.

Lines 2–5 populate the B-LWR model with a number of sample points, each of the form $(\boldsymbol{x}, L_f(\boldsymbol{x}))$, by repeatedly generating random points in the domain, following a RASH trajectory starting from that point (line 4) and storing the result according to the definition (3).

Once the model g is populated, the algorithm proceeds by alternating model minimizations and objective function minimizations (lines 6–9). A promising starting point can be found by minimizing g with a multiple-run RASH heuristic starting from a random point (line 7). The point is used to begin the minimization trajectory for f (line 8). Finally, the result of the optimization run (in terms of initial point, best value in trajectory) is stored into g in order to refine it for the next run.

Note that optimization runs aimed at function f are always single: a repeated run would generate a "broken" trajectory where the final optimum has no relationship with the initial point in the trajectory, therefore the model g would become useless. The same concern is not valid for g minimizations.

4 Experimental Results

We compare the performance of Repeated-RASH and M-RASH on the benchmarks shown in Table 1. Rosenbrock is a unimodal function in the domain $[-100, 100]^d$ with a long narrow valley and has a global minimum of zero located at $(1, 1)^d$. Rastrigin is a multimodal function in the domain$[-10, 10]^d$ with huge number of local minima and a global minimum of zero at origin. The Schaffer function is a 2-dimensional maximization function in the range $[-100, 100]^2$ with a lot of valleys surrounding the global maximum of 1 at $(0, 0)$.

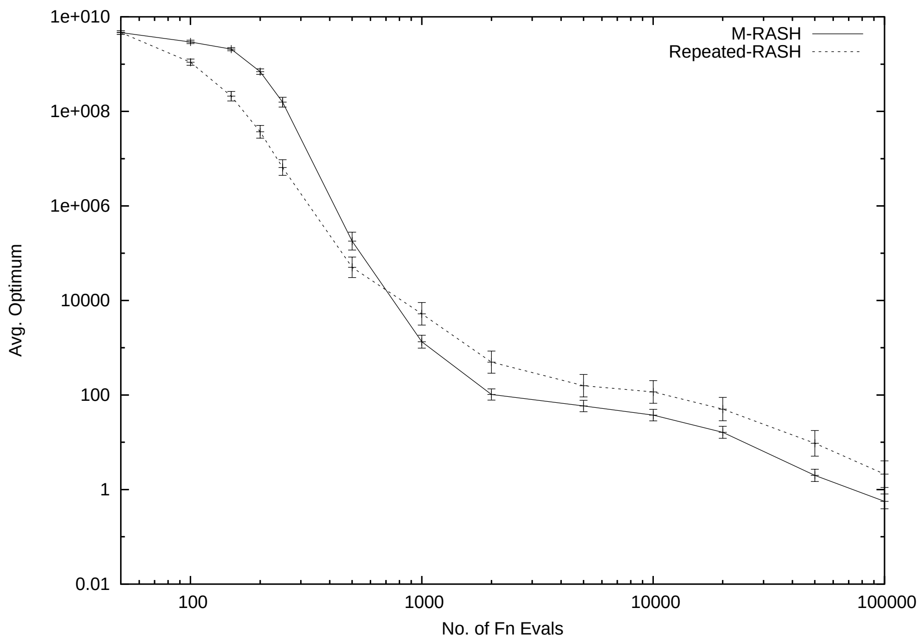

Figure 6: Rosenbrock Function

The termination condition for both Repeated-RASH and M-RASH is set to 100000 function evaluations. In M-RASH, we start with $n = 2$ initial sample points. The termination condition for `RASH`(line 4) in Fig. 5 is set to 50 function evaluations. The idea is to feed the model with couple of sample points before querying it to find the next data points to explore (lines 6–9). The `RepeatedRASH` call(line 7) searches the regression model g for the optimum point. As the execution of `RepeatedRASH` on the model doesn't add to the function evaluations of f, it is run for large number of iterations to make sure that with a high probability that an optimum point on the model is achieved. The call to `RASH`(line 8) takes the optimum point suggested by the `RepeatedRASH` as the starting point $\boldsymbol{x}$ for minimizing f. This call is terminated if RASH fails to improve the optimum value on function f for a fixed number of consecutive steps. Hence, RASH continues to run as long as it is able to find better optimum values and not stuck at local minimum. In our simulations, we terminate the call to RASH(line 8) if it doesn't improve on the optimum value found for 100 consecutive steps. The starting point used by RASH along with the best value found is then added to the regression model g. The above procedure is repeated till the overall termination condition of 100000 function evaluations is met.

The algorithms are run for 100 trails and the average optimum found along with standard deviation is plotted against the number of function evaluations in log-log scale. The comparison graphs between Repeated-RASH and M-RASH are shown in Fig. 6 - 8. The performance of M-RASH is slightly worse at the beginning but even-

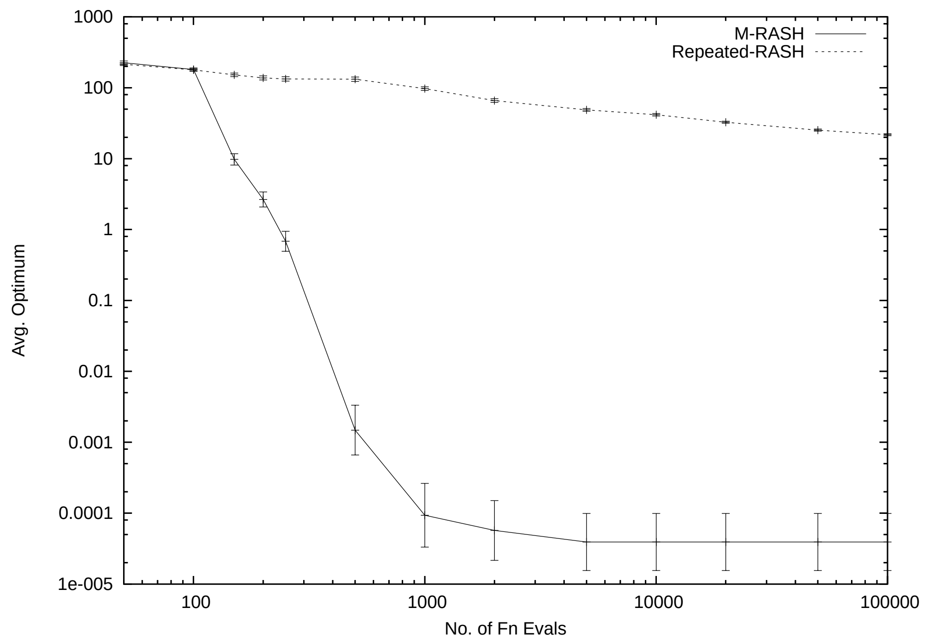

Figure 7: Rastrigin Function

tually better compared to Repeated-RASH for the uni-modal Rosenbrock function as shown in Fig. 6. The M-RASH algorithm outperforms Repeated-RASH for the two multimodal functions as shown in Fig. 7 and Fig. 8. This can be explained by the structure of one-dimensional Rastrigin shown in Fig. 4. Once the B-LWR model is fed with enough sample points to get a global structure of the function, it will immediately direct the local search algorithm to the data points near the global minimum. In the case of Repeated-RASH, due to large number of local minima it often gets stuck at them and thus proceeds slowly towards the global minimum. This is also true for the Schaffer function. Thus, M-RASH quickly converges to the areas close to the global minimum for the functions with high local minima where the B-LWR plays an important role in learning the trend of local minimum and guiding the local search RASH technique to promising areas.

5 Conclusion

We have presented the framework of M-RASH technique with some preliminary results. M-RASH, which is an integration of B-LWR and RASH techniques, results in faster convergence and better average optimum values compared to Repeated-RASH. There are a number of critical parameters in the B-LWR and RASH techniques which include the kernel width K, the kernel function [12], prior assumptions on $\boldsymbol{\beta}$ distribu-

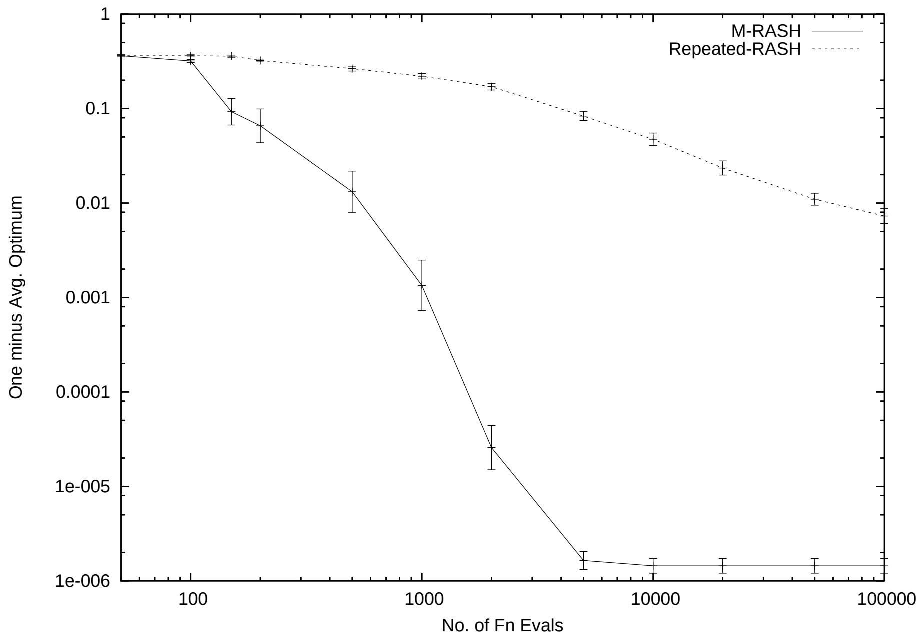

Figure 8: Schaffer Function

tion, the initial number of sample points n, the termination conditions of the function calls `RASH` and `RepeatedRASH` and the initial search region $\mathcal{R}$ in `RASH` algorithm. Ongoing work not described in this paper because of limited space and future efforts will consider the detailed effect of these parameters on the effectiveness of the technique.

References

[1] Roberto Battiti. Partially persistent dynamic sets for history-sensitive heuristics. In D. S. Johnson, M. H. Goldwasser, and C.C. McGeoch, editors, *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Challenges*, volume 59 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–21. American Mathematical Society, 2002.

[2] Roberto Battiti and Mauro Brunato. Reactive search: Machine learning for memory-based heuristics. Technical Report DIT-05-058, Università di Trento, September 2005. To appear as a chapter in the book: Teofilo F. Gonzalez (Ed.), Approximation Algorithms and Metaheuristics, Taylor & Francis Books (CRC Press), 2006.

[3] Roberto Battiti and Marco Protasi. Reactive search, a history-sensitive heuristic for MAX-SAT. *ACM Journal of Experimental Algorithmics*, 2, 1997.

[4] Mauro Brunato and Roberto Battiti. The reactive affine shaker: a building block for minimizing functions of continuous variables. Technical Report DIT-06-012, Università di Trento, February 2006.

[5] W. S. Cleveland and S. J. Devlin. Locally-weighted regression: An approach to regression analysis by local fitting. *journal of the American Statistical Association*, 83: 596–610, 1988.

[6] J. C. Culberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, August 1998.

[7] Artur Dubrawski and Jeff Schneider. Memory based stochastic optimization for validation and tuning of function approximators. In *Conference on AI and Statistics*, 1997.

[8] H. H. Hoos and T. Stuetzle. *Stochastic local search: Foundations and applications*. Morgan Kaufmann, 2005.

[9] W. Jacquet, B. Truyen, P. de Groen, I. Lemahieu, and J. Cornelis. Global optimization in inverse problems: A comparison of kriging and radial basis functions. 2005.

[10] Richard E. Korf and Ariel Felner. Disjoint pattern database heuristics. *Artif. Intell.*, 134(1-2):9–22, 2002.

[11] Andrew Moore and Jeff Schneider. Memory-based stochastic optimization. In D. Touretzky, M. Mozer, and M. Hasselm, editors, *Neural Information Processing Systems 8*, volume 8, pages 1066–1072. MIT Press, 1996.

[12] Andrew Moore, Jeff Schneider, and Kan Deng. Efficient locally weighted polynomial regression predictions. In D. Fisher, editor, *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 236–244, 340 Pine Street, 6th Fl., San Francisco, CA 94104, 1997. Morgan Kaufmann.

[13] Francisco J. Solis and Roger J.-B. Wets. Minimization by random search techniques. volume 6, pages 19–30, 1981.

[14] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.