



Universidad Autónoma  
de Madrid

**Biblos-e Archivo**  
Repositorio Institucional UAM

**Repositorio Institucional de la Universidad Autónoma de Madrid**

<https://repositorio.uam.es>

Esta es la **versión de autor** del artículo publicado en:  
This is an **author produced version** of a paper published in:

Modified Grid Searches for Hyper-Parameter Optimization. In: de la Cal, E.A., Villar Flecha, J.R., Quintián, H., Corchado, E. (eds) Hybrid Artificial Intelligent Systems. HAIS 2020. Lecture Notes in Computer Science 12344 (2020): 221-232

**DOI:** [http://doi.org/10.1007/978-3-030-61705-9\\_19](http://doi.org/10.1007/978-3-030-61705-9_19)

**Copyright:** © Springer Nature

This version of the article has been accepted for publication, after peer review (when applicable) and is subject to Springer Nature's AM terms of use, but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at:

[http://doi.org/10.1007/978-3-030-61705-9\\_19](http://doi.org/10.1007/978-3-030-61705-9_19)

# Modified Grid Searches for Hyper-Parameter Optimization

David López<sup>1</sup>, Carlos M. Alaíz<sup>1</sup>, and José R. Dorronsoro<sup>1,2</sup>

<sup>1</sup> Dept. Computer Engineering, Universidad Autónoma de Madrid

<sup>2</sup> Inst. Ing. Conocimiento, Universidad Autónoma de Madrid

david.lopezramos@estudiante.uam.es, carlos.alaiz@uam.es,  
jose.dorronsoro@uam.es

**Abstract.** Black-box optimization aims to find the optimum of an unknown function only by evaluating it over different points in the space. An important application of black-box optimization in Machine Learning is the computationally expensive tuning of the hyper-parameters, which requires to try different configurations and measure the validation error over each of them to select the best configuration. In this work two alternatives to classical Grid Search are proposed, trying to alleviate the low effective dimensionality problem, i.e., the drop of performance when only some of the dimensions of the space are important. The first approach is based on a modification of a regular grid to guarantee equidistant projections of the points over each axis, whereas the second approach also guarantees these spread projections but with a random component. As shown experimentally, both approaches beat Grid Search, although in these experiments their performance is not statistically different from that of Random Search.

**Keywords:** Hyper-Parameter Optimization · Grid Search · Random Search · Black-Box Optimization.

## 1 Introduction

Optimization problems are ubiquitous in many fields, and they particularly important in Machine Learning, where for example the training of a learning machine is often done by solving an optimization problem, usually with a known objective function. Moreover, the selection of the hyper-parameters is an optimization problem, where the best parameters are selected according to a certain criteria, usually represented by an unknown (or intractable) objective function. Hence, this task is often tackled as a black-box optimization problem, where different configurations, selected using different strategies, are evaluated to find the best one.

Black-box optimization consists in optimizing a certain functional  $F(\mathbf{x})$ , with  $\mathbf{x} \in \mathcal{S}$  (usually,  $\mathcal{S} \subset \mathbb{R}^d$  and hence there are  $d$  optimization variables), when the analytical form of  $F$  may not be available but it can be evaluated at different points. There are multiple black-box optimization algorithms, but two of the

main representatives in Machine Learning, besides the Bayesian approaches, are Grid Search (GS) and Random Search (RS). GS performs an exhaustive search over a discretization of the search space. RS is a simple powerful approach that generates randomly the points to be evaluated and it can outperform GS in searches with a low effective dimensionality, as discussed below. Here, two different approaches are proposed to cope with this problem, trying to guarantee a good performance even if only a small number of the optimization variables of the problem are significant, since these methods are designed to provide samples whose projections over each dimension are equidistant over all the search range. In particular, the contributions of this work can be summarized as follows:

1. A new black-box optimization method, called Modified Grid Search, is proposed to alleviate the low effective dimensionality problem of GS.
2. Another method is proposed, namely Latin Hyper-Cube Grid Search, which is a random version of the previous one.
3. Both methods are compared experimentally with GS and RS.

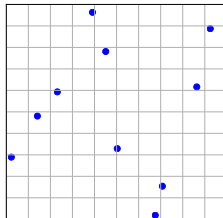
The remaining of the paper is organized as follows. Section 2 contains an overview of hyper-parameter optimization, in particular using standard GS and RS. Section 3 presents the two proposed methods to alleviate the limitations of GS, which are numerically compared with GS and RS in Sect. 4. The paper ends with some conclusions and pointers to further work in 5.

## 2 Hyper-Parameter Optimization

In order to train a parametric learning machine, two different types of parameters have to be set: the internal parameters of the model and the hyper-parameters. The first ones are adjusted from the data during the training phase. But the hyper-parameters usually cannot be learned directly from the data, because they represent properties of a higher level in the model, such as its complexity or its architecture. Therefore, hyper-parameters are usually predefined during a previous validation phase by simply trying different configurations to see which ones work better. Hence, in this phase the quality of the model is seen as a black-box function, in which different configurations of hyper-parameters are inserted and the output is observed to select the configuration with a better performance. Usually, this process is an expensive task since the search space can be large, and evaluating each configuration is costly [5].

### 2.1 Grid Search and Random Search

Two of the most popular methods to optimize hyper-parameters (without considering Bayesian approaches) are the classical Grid Search (GS), and the Random Search (RS) proposed by Bergstra and Bengio [1], who attribute the success of both techniques mainly to their conceptual simplicity, ease of implementation, trivial parallelization, and outperformance over purely manual optimization. Moreover, they show that RS provides a slight reduction of efficiency in low



**Fig. 1.** Latin Hyper-Cube Sampling in 2D.

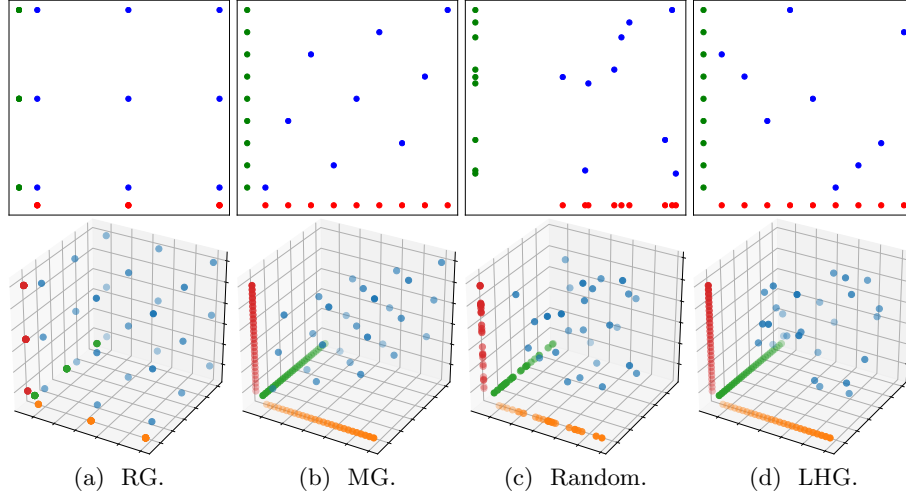
dimensions to obtain a large increase of efficiency in high dimensions with respect to GS, because frequently the objective function has a low effective dimensionality, that is, the function is more sensitive to changes in some dimensions than in others. In particular, it can be the case that for a certain model and a particular dataset, only certain hyper-parameters are relevant. The problem is that the relevant hyper-parameters change from one dataset to another, preventing from a design of an appropriate custom grid even for a fixed model. In this context, the low effective dimensionality causes GS to evaluate points in dimensions that have very little relevance, obtaining a worse performance than RS. Furthermore, RS presents other practical advantages: the experiments can be stopped at any time and the trials form a complete experiment, new trials can be added to an experiment without having to adjust the grid, and the failing of a trial does not jeopardize the complete search (it can be simply discarded).

## 2.2 Latin Hyper-Cube Sampling

A different approach can be found in the sampling method known as Latin Hyper-Cube Sampling (LHS; [7]). In the two-dimensional case, this method divides the search space into different cells, and randomly determines the cells that will be used to sample in such a way that no row or column is selected more than once. Hence, this selects a random cell, then samples a point inside that cell and discards the row and column corresponding to the cell; the process is repeated until discarding all the rows or columns. The advantage of this sampling is that the projection of the points over each dimension fall in different cells, spreading them over the whole interval (an example is shown in Fig. 1). This idea will be the basis of the new search approaches proposed in this work.

## 3 Modified Grid Searches

The GS approach, based on a Regular Grid (RG) and called for now on Regular Grid Search (RGS), performs particularly bad when the function to be minimized has a low effective dimensionality. For example, even if a certain optimization variable  $x_i$  has no real impact on the objective function  $F$ , by its own nature RGS will evaluate different points,  $\mathbf{x}$  and  $\mathbf{x}'$ , that only differ in that variable, i.e.



**Fig. 2.** Illustration of the samples generated by the four methods for a two-dimensional search (top) and a three-dimensional search (bottom). The samples are represented in blue; their projections over the axis are shown in green, red and orange.

$x_j = x'_j, \forall j \neq i$ , although these points will be essentially equivalent, providing approximately the same objective value,  $F(\mathbf{x}) \approx F(\mathbf{x}')$ . This effort could be used to explore instead other combinations of the effective variables, improving the results. In order to alleviate the limitation of RGS when optimizing problems with low effective dimensionality, two sampling methods are proposed next.

### 3.1 Modified Grid Search

The idea underneath the first new method, called Modified Grid Search (MGS), is to use a different configuration of the points to be evaluated so that none of their coordinates is repeated, i.e., so that each optimization variable never takes the same value twice. Moreover, their projections over each dimension will be equidistant to explore the space as much as possible. With this configuration, even if a certain variable has no impact on the objective function, the values explored by MGS for the remaining variables will be exactly the same whether the insignificant variable is considered or not.

The design of this Modified Grid (MG) is based on that of a standard grid, defined as the Cartesian product of the sets of values to be evaluated for each variable. In this case, these sets are equispaced in linear or logarithmic scale. The process of modifying the RG can be understood intuitively in two dimensions, where it can be imagined as if the RG were stretched, pulling the lower left corner and the upper right corner until the points take equidistant projections between them (this can be seen in Figs. 2a and 2b, top).

The procedure is formalised next. Let  $\mathcal{G}_{\mathbf{n}} \subset \mathbb{R}^d$ , with  $\mathbf{n} = (n_1, n_2, \dots, n_d)$ , be the regular grid that tries  $n_i$  different values for the  $i$ -th optimization variable,

and which is defined as:

$$\mathcal{G}_{\mathbf{n}} = \{(x_1, x_2, \dots, x_d) \in \mathbb{N}^d : 1 \leq x_i \leq n_i, i = 1, \dots, d\}.$$

Although this grid is defined in the region  $\prod_{i=1}^d [1, n_i]$ , it can be easily rescaled or adapted to logarithmic scale. The procedure to convert  $\mathcal{G}_{\mathbf{n}}$  into a MG, denoted by  $\mathcal{G}'_{\mathbf{n}}$ , is as follows. For the  $i$ -th variable, and for each value of that variable,  $x_i \in \{1, \dots, n_i\}$ , every appearance is moved a different quantity so that there are no repeated values. In particular, the  $j$ -th appearance of the  $i$ -th variable  $x_i$  is displaced an amount  $(j-1)d_i$ , where  $d_i = 1/\prod_{j \neq i} n_j$  is the inverse of the number of times that each value of the  $i$ -th variable will be repeated. In this way, the last appearance satisfies:

$$x_i + \frac{\prod_{j \neq i} n_j - 1}{\prod_{j \neq i} n_j} = x_i + 1 - d_i,$$

i.e., it is at a distance  $d_i$  from the following value  $x_i + 1$ , hence producing an equidistant distribution of the points.

The whole procedure is summarized in Alg 1, where **RegularGrid** returns the aforementioned RG  $\mathcal{G}_{\mathbf{n}}$ , and where the function **NormalizeGrid** just normalizes the grid to the range  $\prod_{i=1}^d [0, 1]$  (the grid could be further transformed, e.g. resizing and translating it, or changing some components to logarithmic scale).

### 3.2 Latin Hyper-Cube Grid

The idea behind the LHS can be applied to the creation of a Latin Hyper-Cube Grid (LHG), which is simply a deterministic version of the LHS that guarantees equidistant projections, as the proposed MG, but in this case with a random structure instead of one based on the RG. In this way, the points to be evaluated in this new LHG Search (LHGS) will be distributed randomly over a dense grid without sharing any row or column (or any slice in dimension larger than two).

The procedure to generate the LHG is described next. It should be noted that, in this case, there is no point in selecting the number of different values to be tried per optimization variable, since it will be ignored by the algorithm, which only takes into account the total number of points to be evaluated.

Hence, the algorithm is based on the total size sample  $N$ , and on a vector  $\mathbf{z} = (1, 2, \dots, N) \in \mathbb{N}^N$  which represents the  $N$  different values that all the variables will take. Now, in order to define the samples, it is enough to define the  $N$  values of the  $i$ -th variable as a random permutation of  $\mathbf{z}$ , which guarantees that all the variables will take  $N$  equidistant values, distributed over an  $N \times N$  grid without repeating any value twice for any variable.

This procedure is shown in Alg. 2, where **Permutation**( $\mathbf{z}$ ) returns a random permutation of the vector  $\mathbf{z}$ , and again **NormalizeGrid** normalizes the samples.

### 3.3 Comparative

**Illustration** An illustration of the four grids (RG, MG, random grid and LHG) is shown in Fig. 2, both for the two-dimensional (top) and the three-dimensional

**Algorithm 1:** Modified Grid.

---

**Data:** Number of values by dimension,  $\mathbf{n} \in \mathbb{R}^d$ .  
**Result:** Set of points to be evaluated.

```

for  $i = 1$  to  $d$  do
   $d_i \leftarrow \frac{1}{\prod_{j \neq i} n_j}$  // Separation between consecutive values.
end
 $N \leftarrow \prod_i n_i$  // Number of points in the grid.
 $\mathcal{G}_n \leftarrow \text{RegularGrid}(\mathbf{n}) = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$  // Regular grid.
for  $i = 1$  to  $d$  do
  for  $j = 1$  to  $n_j$  do
     $c_{ij} \leftarrow 0$  // Number of apparition of value  $j$  in variable  $i$ .
  end
end
for  $p = 1$  to  $N$  do
  for  $i = 1$  to  $d$  do
     $\bar{x}_i^{(p)} \leftarrow x_i^{(p)} + (c_{ix_i^{(p)}} - 1) d_i$ 
     $c_{ix_i^{(p)}} \leftarrow c_{ix_i^{(p)}} + 1$ 
  end
end
 $\mathcal{G}'_n \leftarrow \{\bar{\mathbf{x}}^{(1)}, \dots, \bar{\mathbf{x}}^{(N)}\}$ 
return  $\text{NormalizeGrid}(\mathcal{G}'_n)$ 

```

---

(bottom) cases, where the samples are shown in blue, and their projections over the different axis in green, red and orange. Some conclusions can be drawn:

- In the case of RG, the points are distributed in a regular grid, and hence the projections collapse always to only 3 points (the number of values per variable). If the objective functional  $F(x, y, z)$  could be (reasonably) approximated by a function of one variable  $G(x)$ , the sample would be equivalent to one of size 3, although 9 or 27 points would be evaluated in two or three dimensions, respectively.
- For MG, the points that previously had repeated projection values now occupy the entire projection interval, obtaining 9 or 27 different and equidistant values for each one of the two or three variables. In the low dimensional case, where  $F(x, y, z)$  could be approximated by  $G(x)$  the MG will continue to have 9 or 27 different values to explore over the significant variable  $x$ .
- For the random sample, there is no structure, hence the projections of the points do not collapse, but they can leave gaps between them.
- For LHG, the points are randomly distributed on the grid but their projections on the axes are equidistant, hence the proposed LHGS presents the robustness against the low effective dimensionality of MG.

**Pair Distance Distributions** A study of the probability distribution of the distances between each pair of points can give an intuition about how the points

---

**Algorithm 2:** Latin Hyper-Cube Grid.

---

**Data:** Number of values by dimension,  $\mathbf{n} \in \mathbb{R}^d$ .**Result:** Set of points to be evaluated. $N \leftarrow \prod_i n_i$  // Number of points in the sample. $\mathbf{z} \leftarrow (1, 2, \dots, N)$ **for**  $i = 1$  **to**  $d$  **do**     $\mathbf{p}_i \leftarrow \text{Permutation}(\mathbf{z})$ **end****for**  $p = 1$  **to**  $N$  **do**     $\mathbf{x}^{(p)} \leftarrow (p_{1p}, p_{2p}, \dots, p_{dp})$     //  $p_{ip}$  is the  $p$ -th entry of  $\mathbf{p}_i$ .**end** $\mathcal{G}_{\mathbf{n}} \leftarrow \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$ **return**  $\text{NormalizeGrid}(\mathcal{G}_{\mathbf{n}})$ 

---

are placed, and how the shape of the grid affects this configuration. Hence, the following experiment is done. The distance between every pair of points of the grids will be measured for 100 generation of the four grids. Obviously, RG and MG will always produce the same grid, since they are deterministic, but the repetition of the experiment allows to cope with the randomness of the random sample and of LHG. The sample will be generated in dimension three, and its total size (the number of points in the grid) will be  $N = 6^3 = 216$ . In Fig. 3 the distribution is shown for three configuration of the vector  $\mathbf{n}$ : (6, 6, 6), (12, 6, 3) and (36, 3, 2), that is, from more to less square. As it can be seen, as the number of points in each dimension is decompensated, the distances in the RG are separated to the right (the number of greater distances between pairs increases), while the other methods keep practically the same distribution.

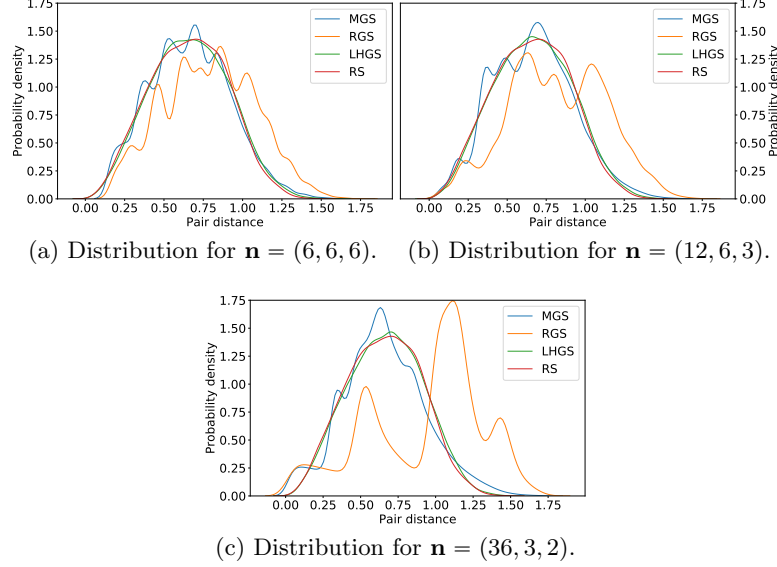
## 4 Experiments

Two different set of experiments are conducted to validate the proposed methods, comparing RGS, MGS, RS and LHGS. The first one consists in the minimization of synthetic (hence, known) objective functions, whereas the second one deals with the tuning of the hyper-parameters of a Support Vector Machine.

### 4.1 Minimum of a Synthetic Function

The first set of experiments just tackles a general black-box optimization problem, aiming to find the minimum value of a function randomly generated. In particular, the function to evaluate will consist of a complete polynomial of a fixed degree  $g$  and dimension  $d$  (i.e., number of optimization variables). The coefficients are generated randomly in  $[-1, 1]$ , and in order to bound the problem from below, the absolute value of the generated polynomial is used as objective functional,  $F(\mathbf{x}) = |p(x_1, x_2, \dots, x_d)|$ .



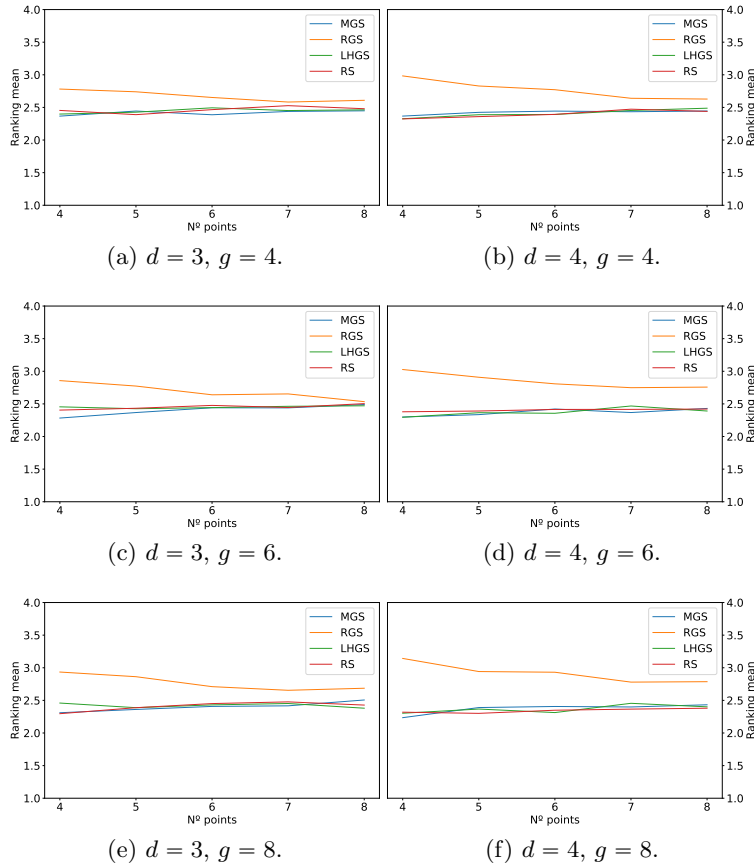


**Fig. 3.** Pair distance distributions for different configurations of the grid.

The sample region (for the four search methods) is scaled to the interval  $[-1, 1]$  in each dimension. The number of values explored by the four methods is the same over all the dimensions (the vector  $\mathbf{n}$  is constant,  $n_i = n$  for  $1 \leq i \leq d$ ).

Three different degrees are used for the polynomial,  $g \in \{4, 6, 8\}$ , and two different number of dimensions,  $d \in \{3, 4\}$ . For a given configuration  $(g, d)$ , 1000 different polynomials are generated to average the results. Different values of  $n$  are used to see the evolution of the search approaches. For each polynomial and for each value of  $n$ , the polynomial is evaluated over the solution given by each method, and the resultant four values are used to build a ranking that will be averaged over the 1000 repetitions.

The results are shown in Fig. 4. It can be seen that, the larger the degree of the polynomials  $g$ , the larger the separation between RGS and the other three methods, which have essentially the same performance. Similarly, the performance of RGS is worse when  $d$  is 4, since in that case, a low effective dimensionality is more probable, although it is not explicitly enforced. These results can be further analysed using a Nemenyi test [8] to compare the average ranks. This test is based on the creation of a threshold above which the difference between two means are considered significant. With the ranking means and the number of iterations, the critical distance is  $D = 0.148$  for a significance value  $\alpha = 5\%$ . If the means of two methods have a greater difference than this value, one of them can be considered better than the other. Looking at the results corresponding to  $g = 6$  and  $g = 8$ , the difference between RGS and the other methods is significant, whereas between the other three methods is not.



**Fig. 4.** Average ranking for the minimization of a synthetic polynomial, for different degrees of the polynomial  $g$  and different numbers of dimensions  $d$ .

As a conclusion, it seems that the performance of RGS is worse the greater the complexity of the polynomial to be evaluated, because raising the number of variables and raising the degree, the number of possible combinations increases, and the other methods make a better use of the available exploration effort. In addition, the performance in this problem of MGS, RS and LHGS is similar. It is necessary to mention that the importance of each variable is given by its (random) coefficient, so an ineffective dimension is not enforced. If some variables would be ignored to build the polynomial, probably the performance of the new methods would be even better when compared to RGS.

## 4.2 SVM Hyper-Parameter Tuning

The following experiments compare the four search approaches when applied to tuning the hyper-parameters of a Support Vector Machine (SVM; [2,9]). These

experiments are implemented in Python, using the *scikit-learn* library. In particular, the `GridSearchCV` function is used, but configuring a list of dictionaries that define the searching space according to RGS, MGS, RS or LHGS. To evaluate each SVM configuration, 5-fold cross-validation is used. Therefore, the black-box optimization problem consists in minimizing the 5-fold cross-validation error, and the optimization variables are the hyper-parameters. The kernel selected for the SVMs is the polynomial kernel  $k(\mathbf{x}, \mathbf{x}') = (\gamma \langle \mathbf{x}, \mathbf{x}' \rangle + \tau)^d$ , because it has more hyper-parameters than the most commonly used Gaussian kernel, and hence it allows to compare the search approaches in higher dimensional spaces (with more optimization variables). There are in total 5 hyper-parameters:

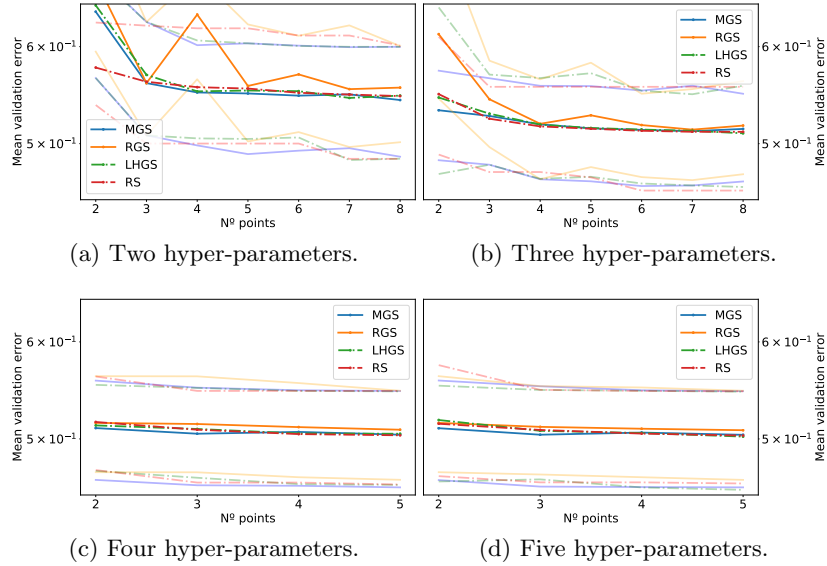
- `C` ( $C$ ): Penalty parameter for the error term.
- `epsilon` ( $\varepsilon$ ): Width of insensitivity.
- `degree` ( $d$ ): Degree of the polynomial kernel.
- `gamma` ( $\gamma$ ): Kernel coefficient.
- `coef0` ( $\tau$ ): Kernel independent term.

The comparison is done over two real (although smallish) regression datasets: *Diabetes* (introduced in [4], and available through the function `load_diabetes`) and *Boston* (corresponding to the UCI repository [3], and accessible through the function `load_boston`). For each dataset, and for each search method, the validation score, e.g., the cross-validation error corresponding to the best hyper-parameters, is used as a measure of the performance of the optimization approach. Four experiments are conducted for each dataset, tuning two, three, four and five hyper-parameters, with the rest fixed to their default values, and the procedure is repeated 10 times with different partitions of the datasets.

The validation errors are shown in Figs. 5 and 6, where the dark lines represent the mean best validation error for each method, and the light lines below and above represent the minimum and maximum values (to show how spread are the results). Looking at the experiments, RGS is worse than the other three methods when the number of evaluated points is small, although the difference is reduced when the grid becomes denser. The other three methods behave similarly, with small differences between them. Finally, the oscillations of RGS should be remarked, since when the number  $n$  of different values per hyper-parameter is small then the difference between considering  $n$  or  $n + 1$  values can change a lot the distribution of the samples in the space, affecting the performance.

## 5 Conclusions

Black-box optimization is an ubiquitous problem, specially important in Machine Learning, where tuning the hyper-parameters of a learning machine is a particular case that can be crucial for guaranteeing the good performance of many methods. Two of the most popular approaches for setting up the hyper-parameters are Grid Search (GS) and Random Search (RS), the first one based on an exhaustive search over a discretisation of the space, and the second one on generating and evaluating random points.



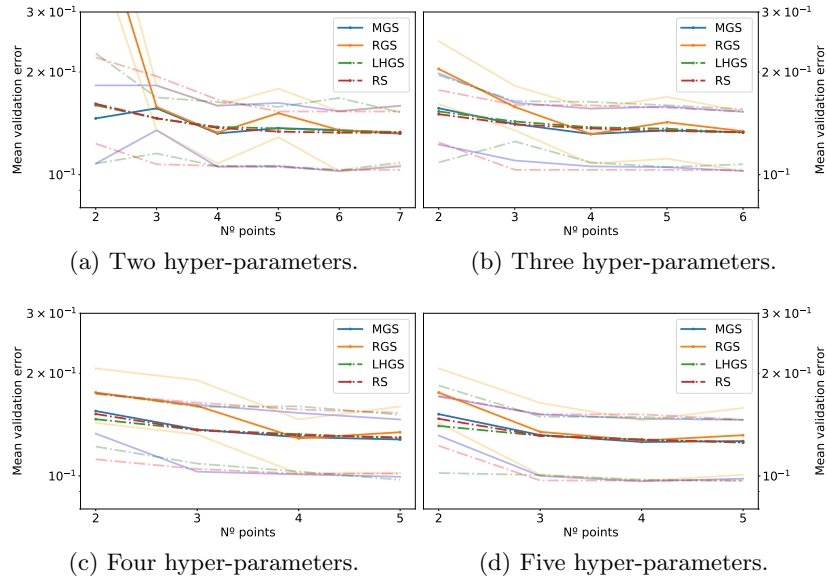
**Fig. 5.** Validation errors for *Diabetes*.

In this work two new approaches are proposed. The first one, Modified Grid Search (MGS), is based on a modification of a regular grid to guarantee the best possible independent exploration of each hyper-parameter, by providing equidistant projections of the grid-points over every axis. The second approach, Latin Hyper-Cube Grid Search (LHGS), is a random version of the first one, that again provides equidistant projections but without enforcing any structure on the grid, similar to a deterministic version of the Latin Hyper-Cube Sampling. As shown experimentally, when considering the same number of explored points, and hence the same computational effort, the proposed methods outperform GS in many situations, being comparable, and in some situations better, than RS.

As further work, the illustrative experiments shown in this paper should be extended to consider a large number of datasets to average the performance, so that significant results can be obtained. There are also many other approaches for hyper-parameter optimization that could also be added to the comparison (e.g. [6]). It should be noted that only the hyper-parameters of a Support Vector Machine with polynomial kernel have been tuned, but it would be interesting to extend this study to models like (possibly Deep) Neural Networks, where the large number of hyper-parameters could make the low effective dimensionality problem more severe, and hence the proposed approaches more beneficial.

### Acknowledgements

With financial support from the European Regional Development Fund and from the Spanish Ministry of Economy, Industry, and Competitiveness - State Re-

**Fig. 6.** Validation errors for *Boston*.

search Agency, project TIN2016-76406-P (AEI/FEDER, UE). Work supported also by the UAM-ADIC Chair for Data Science and Machine Learning.

## References

1. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. *Journal of Machine Learning Research* **13**(Feb), 281–305 (2012)
2. Cortes, C., Vapnik, V.: Support-vector networks. *Machine learning* **20**(3), 273–297 (1995)
3. Dua, D., Graff, C.: UCI machine learning repository (2017)
4. Efron, B., Hastie, T., Johnstone, I., Tibshirani, R., et al.: Least angle regression. *The Annals of statistics* **32**(2), 407–499 (2004)
5. Feurer, M., Hutter, F.: Hyperparameter optimization. In: *Automated Machine Learning*, pp. 3–33. Springer (2019)
6. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research* **18**(1), 6765–6816 (2017)
7. McKay, M.D., Beckman, R.J., Conover, W.J.: Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics* **21**(2), 239–245 (1979)
8. Nemenyi, P.: Distribution-free multiple comparisons. In: *Biometrics*. vol. 18, p. 263. International Biometric Soc 1441 I ST, NW, SUITE 700, WASHINGTON, DC 20005-2210 (1962)
9. Smola, A.J., Schölkopf, B.: A tutorial on support vector regression. *Statistics and computing* **14**(3), 199–222 (2004)