Initialization of Bayesian Optimization Viewed as Part of a Larger Algorithm Portfolio

Marius Tudor Morar
School of Computer Science
University of Manchester
M13 9PL, United Kingdom
Email: mtm@cs.man.ac.uk

Joshua Knowles
School of Computer Science
University of Birmingham
B15 2TT, United Kingdom
Email: j.knowles@cs.bham.ac.uk

Sandra Sampaio
School of Computer Science
University of Manchester
M13 9PL, United Kingdom
Email: s.sampaio@manchester.ac.uk

***Abstract*—We consider the problem of setting the hyperparameters of one or more surrogate-assisted evolutionary optimization algorithms, and similar methods such as Bayesian Optimization (i.e. Gaussian Process regression combined with an acquisition function for choosing next solutions to sample), which are often used for problems with expensive function evaluations. It has been remarked elsewhere that these algorithms can be started with an initial experimental design, or with a random sample, or by starting from only two points. We here investigate how to make such choices. By equating the use of random search for the initial population (or design) with *running* a single-point random searcher for some few initial samples (and extending this view to other initialization methods) it seems clear that these methods (initialization + Bayesian optimizer) are rather like an algorithm portfolio, and something can be learned from that literature. However, we start largely afresh with experiments that combine different sampling methods (random search, Latin hypercube) with Gaussian processes and different acquisition functions (expected improvement, and a generalisation of it). We consider a number of different experimental setups (functions and dimensions), and attempt to get a rough view of what works well where. Our work complements some previous Evolutionary Computation work on initialization using subrandom sequences, and experimental designs, but considers more modern algorithms for expensive problems.**

I. Introduction

Recent trends in Bayesian optimization play down the importance of an initial sampling plan, starting with only one or two points. High profile examples are Spearmint [18] and SMAC [6]. But this wasn't always the case: the well known EGO algorithm [9] starts with an initial sampling plan (or space filling design), specifically a Latin hypercube design of approximately $10d$ points, d being the number of dimensions of the search space.

In our work we test these claims under a different perspective, that of an algorithm portfolio. We look at space filling designs as optimization algorithms themselves, and make a more general claim: that different algorithms are better suited to different parts of the optimization process. This extends beyond using just two algorithms: it is perfectly possible, and potentially useful, to select three or more algorithms for different stages. As an example, using random search or a Latin hypercube at the initial stage, using SMAC in the intermediate stage, and Spearmint in the final stage. Of course, the more algorithms are chained together the more possible switch points that must be selected, and we would have on our hands a large hyperparameter optimization problem that may be unnecessarily complicated and challenging to solve effectively.

Traditional algorithm portfolios consist of a set of algorithms from which we choose at each step. We contend that this presents an overly large number of degrees of freedom in the portfolio, since we can be fairly sure that exploration is needed more at the start, exploitation more at the end, and can identify a priori an ordering of switching between algorithms. This reduces the need for hyperparameter optimization (referred to above) quite considerably.

II. Background

We differentiate between two kinds of optimization algorithms: sequential and non-sequential (or nonadaptive). Whereas sequential algorithms base the decision of where to sample next on the results obtained in the previous samples, non-sequential algorithms set out a sampling plan in advance and proceed with it. Non-sequential optimization algorithms generally assume no prior knowledge of the search space and attempt to sample the whole space evenly, in a purely exploratory fashion. Because of this, they are generally known as space filling designs. We recommend [21] for a review of some of the methods of each type.

A. Space filling designs

1) Random search: Random search is possibly the simplest optimization heuristic used, but nevertheless performs well in some scenarios [2]. As it can be intuited, the whole algorithm consists of sampling random configurations and returning the best one found. Because of its simplicity and its usefulness in some tasks, it is commonly considered as a baseline when assessing the performance of other algorithms.

2) Latin hypercube design: Latin hypercube designs attempt to address a commonly perceived weakness of random search, the possibility of several samples being clustered together in a small space, while leaving other portions of the search space less explored. It works by dividing each dimension of the search space into equal parts and assigning samples such that there is a sample in each part of each dimension. An example of this can be seen in figure 1. Many

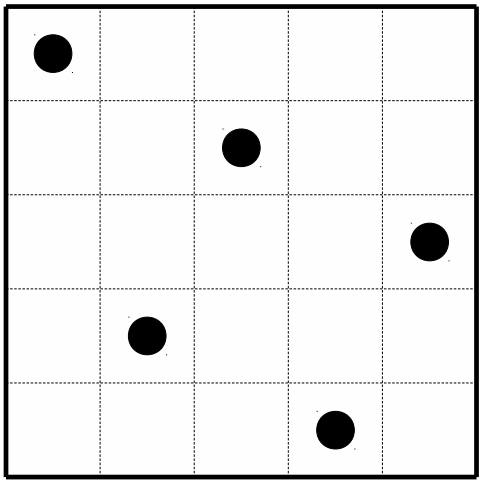

Fig. 1. A five point Latin hypercube sample plan in two dimensions

high dimensional optimization problems have a low effective dimensionality, with only some dimensions actually having an effect on the outcome. One advantage of Latin hypercubes in a problem of this type is that they guarantee that each dimension is sampled across its whole range.

B. Sequential model-based optimization

Sequential algorithms take into consideration the points sampled so far when selecting a new candidate point. Established techniques generally can be split into model-free (e.g. evolutionary algorithms, particle swarm optimization) and model-based (e.g. EGO [9], Spearmint [18], SMAC [6], SPO [1]). Model-based optimization (mostly synonymous with Bayesian optimization) follows a specific pattern for each iteration[1]:

1) create a prediction model M based on currently sampled points;
2) find the best point(s) to sample next by maximizing an acquisition function, which itself is based on model M;
3) sample the new point(s);
4) repeat from the first step unless a stopping criterion is met — commonly just a limit on the number of iterations.

Different components can be put together to form an algorithm; we discuss some common choices. For longer expositions of than we can afford here, we refer the reader to [5], [17].

1) Gaussian process (GP) model: Optimization viewed from a Bayesian framework assumes a prior distribution over the search space. Gaussian process regression models are one of the most commonly used prior distributions in Bayesian optimization. They represent an extension of the multivariate Gaussian distribution to infinite dimensionality, in the sense that each point in the search space is treated as a random variable, or one dimension of the Gaussian distribution. A spatial covariance function between points is defined in order to fully specify the Gaussian process. For details on how to select a suitable covariance function and generally more details on Gaussian processes we refer the reader to [14], likely the best known book on Gaussian processes.

2) Random forest model: Random forests [4] are a well known model that works by aggregating the result of many decision trees, with various forms of randomness introduced in the training of the individual trees. In order to address the exploration versus exploitation tradeoff during the optimization, the model used should output both a prediction, as well as the uncertainty of its prediction (in the form of a confidence interval). While random forests were not designed with this aim, the authors of SMAC use the empirical mean of the individual trees to achieve the same effect [6], which works well in practice, even though it is a heuristic with no theoretical backing.

3) Acquisition functions: The acquisition function (also known as the infill criterion) is a quantity we define that measures the usefulness of sampling a certain point next. By this definition, the next point we should sample is the one that maximizes the acquisition function.

One of the most successful acquisition functions used so far is the expected improvement (EI) [11], which generally achieves a good balance between exploration and exploitation. Assuming a minimization problem, the improvement that is brought by sampling a new point is defined as $I[\theta] = \max(0, f(\theta_{best}) - f(\theta))$, where $f(\theta_{best})$ is the best function value found so far and $f(\theta)$ is the value of f at the point we're sampling next. However, the value of $f(\theta)$ is unknown until we actually sample, but we can integrate over all values of f. The expected improvement then becomes

$$EI[\theta] = E[I[\theta]] = \int_{-\infty}^{f(\theta_{best})} xp(f(\theta) = x)dx.$$

A generalization of the expected improvement: $E[I^g[\theta]]$ is defined in [16], for which higher values of the parameter $g \in \{1, 2, 3, ...\}$ make the optimization more exploratory. In [6], [7] a variation of the expected improvement is used that takes into account the log-transformation of the output values.

One of the older acquisition functions used is the probability of improvement (PI) [10], which favours local exploitation over exploration if the target is unknown [8]. Another acquisition function we mention is the upper confidence bound (UCB) [19], which originates from the multi-armed bandit literature.

4) Maximization of the acquisition function: Acquisition functions provide us with a measure of the benefit of sampling a specific point, but we are left with finding the sample that maximizes this quantity — an optimization problem in itself. Fortunately, most of the time the acquisition function is available as a closed-form solution, which is cheap to evaluate by comparison to the original problem. Therefore, the choice of optimizer for this stage is largely problem dependent, with

[1]Some algorithms include additional steps in order to deal with specialized problem constraints (like noisy measurements), and some don't exactly respect this pattern (e.g. TPE [3]), but our description is kept general enough that it describes most model-based algorithms.

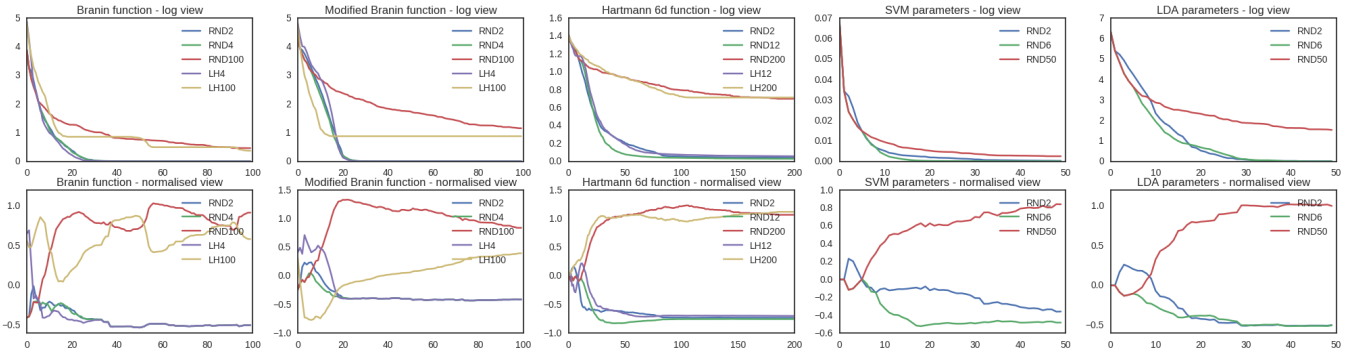

Fig. 2. We compare the results of several optimisers over five test functions. Branin, Modified Branin, and Hartmann 6d are functions in the continuous space, while SVM and LDA parameter optimisations are done over a preset grid. All optimisers sample a number of points before proceeding with the $GP + EI$ aproach. We compare optimisers that sample 2 points randomly (denoted RND2), $2d$ points randomly, where d is the nubmer of dimensions in the test function (denoted RND4, RND6, or RND12), and that sample $2d$ points in a Latin hypercube (denoted LH4, or LH12). We also included random search and a Latin hypercube-only search for comparison (denoted RND50, RND100, RND200, LH50, LH100, LH200). Note that we did not test Latin hypercube methods in the grid-based test functions, as to our knowledge there is no way to apply this. All tests are averaged over 100 runs.

some reasonable choices being evolutionary algorithms, grid search, multi-start local or multi-start gradient search.

III. Hypotheses

Given conflicting claims in the literature, our first aim is to shed some light on the topic of initialisation in Bayesian optimisation. We test the necessity of the initial sampling plan, as well as the specifics of the initialisation method (i.e. random vs Latin hypercube). We state that at least some form of initial sampling (greater than one or two samples) is necessary, for the simple reason that a model built on one or two initial points can't predict anything meaningful. Specifically, the number of points in the initial design is expressed in terms of the number of dimensions or the total sampling budget. As a bare minimum, we suggest using $d+1$ samples, with d being the number of dimensions — any less and not even a linear regression over the points is well defined, so attempting to fit a nonlinear model would make even less sense. A larger initial sample might be more appropriate, but this depends entirely on the problem.

We take the opportunity to note here the difference in opinion on the topic of initialisation. On page 149 of [15]:: *It has not been demonstrated that LHDs are superior to any designs other than simple random sampling (and they are only superior to simple random sampling in some cases).* Agreeing with this is [13]: *The first conclusion of this benchmark analysis is the limited influence of the initial design size on the optimisation results compared to other parameters.* But this is contradicted by [1]: *The experimental analysis clearly demonstrated that the determination of a suitable initial design is of crucial importance for the second phase, which performs a local tuning.* We run tests using $2d$ points as the initial sample, slightly more than the bare minimum we suggest, but still considerably less than the $10d$ initial sampling of EGO.

We make the assertion that the balance between exploration and exploitation should not be kept constant throughout the whole optimisation process, but that more exploration is required in the initial stages than the final stages of optimisation, and the other way around for exploitation. Without loss of generality, a good strategy would be to cover most of the search space in the initial iterations, and then proceed to refine them. Otherwise, we run into the risk of prematurely refining substandard solutions, thus wasting valuable iterations.

To give an example, we look at the expected improvement criterion (EI). From a theoretical perspective EI is myopic: it is ideal if there is only one more point to be sampled, but otherwise it is less than optimal. Multi-step lookahead is possible with EI, and theoretically sound, but does not have a closed form solution, which makes the computation of even two-step lookahead computationally demanding. We direct the reader to [12] for a further discussion on the topic, with the mention that the authors discuss a Bayesian expected loss criterion, although the same holds for EI, as the two are similar.

Since the first part of an optimisation routine is the initial sampling, which is purely exploratory, and the second part is some form of strategy that balances exploration and exploitation, it makes sense to think of the whole optimisation as a multi-step process, or a portfolio of successive algorithms. We investigate using a succession of three algorithms: the initial design, a more exploratory version of EI, and finally the standard EI.

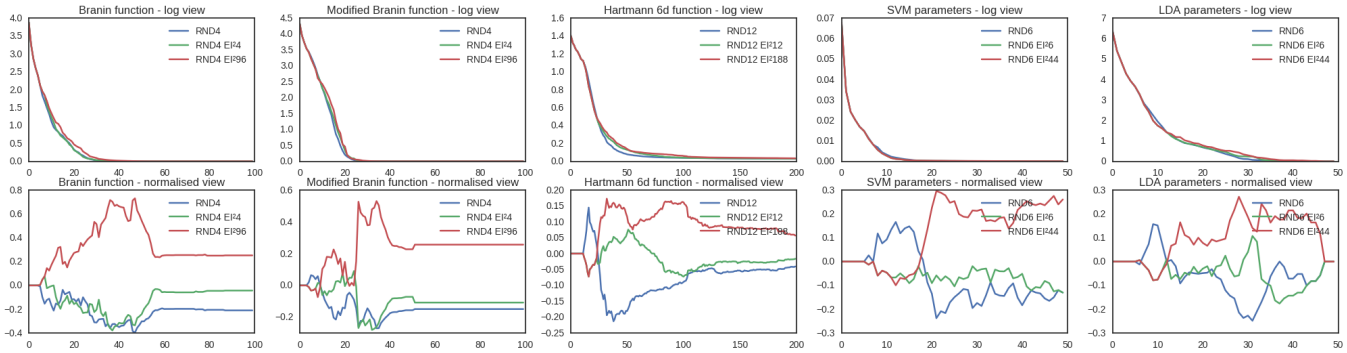

Fig. 3. We tested three optimisers over the same functions and same methodology as in figure 2. All the optimisers tested start with a random sample of size $2d$, where d is the number of dimensions of the test function. Following that, one optimiser proceeds with $GP + EI$ for the rest of the iterations (denoted RND4, RND6, or RND12), another with $GP + E[I^2]$ for $2d$ iterations, then $GP + EI$ for the rest (denoted RND4 EI^{2}4, RND6 EI^{2}6, or RND12 EI^{2}12), and the third one with $GP + E[I^2]$ for the rest of the iterations (denoted RND4 EI^{2}96, RND6 EI^{2}44, or RND12 EI^{2}188). All tests are averaged over 100 runs.

IV. Experiments

A. Initial tests

The first set of tests is meant to establish the importance of the initial sampling plan. The sequential optimisation part is based on a Gaussian process model with expected improvement. For the initial sample we test two random points and a sample of size $2d$, the latter chosen either at random or using a Latin hypercube. As base comparisons, we add random search and Latin hypercube sampling without any sequential algorithm part, for a total of five algorithms compared.

As test functions we use Branin (two-dimensional, also tested in EGO), modified Branin[2] (two-dimensional), Hartmann 6 (six-dimensional, also tested in EGO), and parameters for SVM and LDA on grids of precomputed values (also tested in Spearmint). All tests are repeated with 100 different random seeds.

B. Algorithm portfolios

The next series of tests will be done with a constant initialisation method, and varying the acquisition function. This will be either EI, E[I^2], or a succession of the two: E[I^2] for $2d$ iterations, then EI for the rest. We run E[I^2] before EI as it is more explorative, and as we have stated before, the initial iterations should be more focused on exploration than the last ones.

V. Results

It is important to state from the beginning that making a useful comparison between optimisers is hard. Not only are differences small in terms of absolute value, but the optimiser ranking varies with the budget (i.e. one might be better if we can only afford a small number of function evaluations, but another for longer runs).

We denote by $S_{i,r}$ the function values sampled up to iteration i for a run with random seed r, where $1 \leq i \leq I$ (the optimisation budget) and $r \in R$ (all the seeds used). The most common way of testing optimisation algorithms is to run each algorithm several times with a different random initialisation, and take the best value found up to each iteration and compare the average over the different runs:

$$\frac{1}{|R|} \sum_{r \in R} best(S_{(I-1),r})$$

As we will see from the plots, this approach is at best incomplete. One reason is that different optimisers might find the same optimum, but might not converge at the same speed, which is often important. Another situation might arise in which optimiser A finds better values in the first few iterations than optimiser B, but by the end of the run is surpassed by B. In this case, declaring one to be better depends on the budget of our task.

We look at two different types of plots — the log of the average best value, and the per-iteration normalised value (to 0 mean and 1 standard deviation). We do not show the absolute value of the function, as the differences between different optimisers are too close to be of any vaule. The differences are still small in log space, but are visible. By centering and dividing by the per-iteration average, however, we manage to highlight the differences between similar results. In order not to clutter the plots too much, we create two plots that compare between different optimisers.

The first series of tests compares methods of choosing the initial sample: 2 random samples, $2d$ random samples, d being the number of dimensions, and a Latin hypercube of size $2d$. The results are shown in figure 2. Random search and

[2]Branin$_{mod}(x, y)$ = Branin$(x, y) - 5x$; Branin has three global minima, whereas this variation has one global and two local minima.

Latin hypercube-only search are also included for comparison. We will state the obvious first: random search and Latin-hypercube search are ineffective in all cases except when dealing with a very small optimisation budget. One curious result is the Latin hypercube search over the modified Branin function, which for the inital 10 iterations seems to converge significantly faster than the other optimisers, but does not manage to find any better solutions after that. Overall, the differences between the different initialisation strategies are small, but some differences can be noticed from the plots: in the Hartmann function and for SVM parameter optimisation, the random sample of size $2d$ gives the best results, while at the same time not being worse on the other benchmarks. Latin hypercube initialisation of size $2d$ seems to be marginally better for the Branin function, but slightly worse than the other two initialisation methods for the modified Branin and the Hartmann functions. Note that Latin hypercubes can't be applied over a grid.

The second series of tests sets a fixed initialisation of $2d$ random points, and varies the objective function. We compare the usage of EI exclusively, E[I^2] exclusively, and E[I^2] for the first $2d$ iterations (after initialisation) with EI for the final iterations. The results can be seen in figure 3. From our tests it seems like using standard EI leads to the best results. Using E[I^2] for $2d$ iterations, followed by EI performs almost as good as plain EI, with the Hartmann function being the only test function in which it performs slightly worse. Using E[I^2] exclusively performs worse than either of the previous two methods. It is entirely possible that for a function that is less regular E[I^2] might be more efficient than EI, but so far our limited number of tests fail to show that.

If a specific type of problem is to be optimised many times, we recommend fine tuning the hyperparameters of the optimiser itself, like the initial sample or the acquisition function portfolio. If, however, one only wishes to optimise a problem once, some sensible defaults should be suggested. In this sense, using a random sample of size $2d$ seems reasonable, followed by a Gaussian process and Expected Improvement-based procedure. Using E[I^2] for a few iterations followed by EI might give slightly better results on some problems, but considering that in our tests the difference was small, for the sake of simplicity we recommend using only EI.

VI. Conclusion

The well-known *no free lunch theorem in optimisation* [20] states that averaged over the space of all optimisation problems all algorithms perform the same. Our tests show similar results, with the best optimisation method varying from problem to problem. This suggests we should focus on two goals when developing new methods: first, the optimisers should be flexible enough to allow careful tuning to the a specific problem when needed; second, they should have sensible defaults, that work reasonably well across all of the problems of interest.

In the context of the first goal, we suggesting having a wide variety of interchangeable components (i.e. initial sampling strategies, models, acquisition functions), and selecting the best ones to use for each problem, while at the same time tuning optimiser hyperparameter (e.g. initial sample size). Since we are interested in optimisation, we should not be scared of optimising the hyperparameters of our own algorithms. In the context of the second goal, having sensible defaults is desirable, such that the optimiser can be applied out-of-the-box on a new problem. In the future it would be interesting to investigate methods of predicting when to switch over from one stage to the next, if only to make the whole optimisation process more simple and efficient.

While our work tests the chaining of multiple algorithms over the course of an optimisation routine, more experiments would be beneficial to a better understanding. In particular, we mention testing on the hyperparameter optimisation of deep learning methods as seen in [18] and of integer programming and SAT solvers as in [6].

References

[1] Thomas Bartz-Beielstein and Mike Preuss. Considerations of budget allocation for sequential parameter optimization (SPO). In *Workshop on Empirical Methods for the Analysis of Algorithms, Proceedings*, pages 35–40, 2006.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[3] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[4] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] Alexander IJ Forrester and Andy J Keane. Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences*, 45(1):50–79, 2009.

[6] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[7] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Kevin P Murphy. An experimental investigation of model-based parameter optimisation: SPO and beyond. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 271–278. ACM, 2009.

[8] Donald R Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, 21(4):345–383, 2001.

[9] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[10] Harold J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106, 1964.

[11] J Mockus, V Tiesis, and A Zilinskas. Toward global optimization, volume 2, chapter bayesian methods for seeking the extremum. 1978.

[12] Michael A Osborne, Roman Garnett, and Stephen J Roberts. Gaussian processes for global optimization. In *3rd international conference on learning and intelligent optimization (LION3)*, pages 1–15. Citeseer, 2009.

[13] Victor Picheny, Tobias Wagner, and David Ginsbourger. A benchmark of kriging-based infill criteria for noisy optimization. *Structural and Multidisciplinary Optimization*, 48(3):607–626, 2013.

[14] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[15] Jerome Sacks, William J Welch, Toby J Mitchell, and Henry P Wynn. Design and analysis of computer experiments. *Statistical science*, pages 409–423, 1989.

[16] Matthias Schonlau, William J Welch, and Donald R Jones. Global versus local search in constrained optimization of computer models. *Lecture Notes-Monograph Series*, pages 11–25, 1998.

[17] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[18] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[19] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

[20] David H Wolpert, William G Macready, et al. No free lunch theorems for search. Technical report, Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[21] Antanas Žilinskas and Anatoly Zhigljavsky. Stochastic global optimization: A review on the occasion of 25 years of informatica. *Informatica*, 27(2):229–256, 2016.