A Summary and Review of "Sequential Optimization through Locally Important Dimensions (SOLID)"

Miles Woollacott

Spring 2025

1 Introduction

A common objective in a computer experiments setting is to maximize some multidimensional function f. Unfortunately, more traditional optimization methods such as EGO [1] and active learning McKay [2] may be too slow to implement for more expensive functions. The slower nature of f necessitates the consideration of sequential algorithms. At each iteration, the algorithm needs to calculate a surrogate $\hat{f}$, an estimated maximum $\hat{\boldsymbol{\chi}}$, and an acquisition function, which then determines the location of the next point to be sampled.

However, sequential algorithms require us to maximize $\hat{f}$ and the acquisition function at each iteration, which still may take a long time to evaluate. When this is the case, it may instead be prudent to consider optimizing f, $\hat{f}$, and the acquisition function over a subset of the variables, in order to reduce the number of iterations needed to find the maximum, denoted as $\boldsymbol{\chi} = \arg\max_x f(x)$. The main idea of existing dimension reduction techniques is that optimizing over the entire set of variables may be unnecessary. Indeed, if a variable doesn't contribute much to the function, then the additional time needed to include that variable in the optimization process will compound, all to return a similar result than had we not even considered it. More specific ways to identify such variables will be discussed in Sections 2 and 3.

If f is slow to optimize over, then it makes sense to minimize the number of iterations needed in a sequential learning method, while still returning useful results. First introduced in [3], sequential optimization in locally important dimensions (SOLID) is proposed to satisfy this need by performing multiple variants of variable selection.

The rest of the report is as follows. In Section 2, we provide more detail about the setting of the problem and some methodologies about variable selection. In Section 3, we introduce the idea of local variable selection. In Section 4, SOLID is introduced in detail. In Section 5, we compare SOLID to several competing sequential optimizers in a simulation study, and SOLID is applied to a dataset first introduced in [4]. In Section 6, advantages and disadvantages of SOLID are discussed, and my own commentary for the paper will be provided.

2 Background

We are in a computer experiment setting, and so we consider a Gaussian Process (GP) with noise to estimate f and $\boldsymbol{\chi}$. More precisely, our response vector is $\underline{y} = f(\underline{x}) + \underline{\epsilon}$, where $f(\underline{x}) \sim GP(\mu \underline{1}_p, \Sigma(x, x'))$, and $\underline{\epsilon}_i \overset{\text{iid}}{\sim} \mathcal{N}(0, \tau^2)$ is our noise term. We use the Squared Euclidean correlation matrix

$$\Sigma(x, x') = \sigma^2 K(x, x'), \text{ where } K(x, x') = \exp\left\{ -\sum_{k=1}^{p} \gamma_k (x - x')^2 \right\} \text{ and } \gamma_1, \gamma_2, \ldots, \gamma_p \geq 0.$$

The reader should note that, if $\gamma_k = 0$, then the kth variable does not contribute to $f(\underline{x})$. This will connect to our goal of identifying variables that are necessary to optimize over, and ultimately perform dimensionality reduction. Also note that γ_k is unknown, and so it will be estimated.

The paper notes that a strong the initial design is extremely important for a sequential optimization algorithm. The paper considers a space-filling maximin LHS, since this design should be better at identifying potential regions where $\boldsymbol{\chi}$ may be. I will provide my own opinions on the choice of initial design in Section 6.

SOLID employs both global and local variable selection. We now formally define a method to

perform global variable selection (local variable selection will be covered in Section 3). A variable is called *globally inactive* if $\gamma_k = 0$, and *globally active* if $\gamma_k > 0$. Combining with what was said earlier, this means that globally inactive variables don't contribute anything to $f(\underline{x})$. Therefore, a reasonable idea would be to optimize f is to optimize over all globally active variables. This idea is referred to as the Global Variable Selection (GVS) algorithm. GVS is a sequential optimization algorithm, where at each iteration, we calculate the probability that a variable is globally active by imposing the following priors proposed by [5]:

$$\gamma_k = u_k b_k,\ u_k \sim \text{Gamma}(a_u, b_u),\ b_k \sim \text{Ber}(\theta), \text{ and } \theta \sim \text{Beta}(a_\theta, b_\theta).$$

Note that u_k is independent of b_k. If $b_k = 0$, then u_k is updated via a sliding uniform candidate distribution. More specifically,

$$u_k \sim \text{Unif}\left(\max\{0, u_k - 50\epsilon(u_k)\}, u_k + \epsilon(u_k)\right), \text{ where } \epsilon(u_k) = \begin{cases} \min\{50, u_k h\} & , u_k \geq 30 \\ \max\{1, u_k h\} & , u_k \in [0, 30) \end{cases},$$

where $h \sim \text{Unif}(\frac{1}{2}, 2)$. A discussion of u_k's proposal distribution will be discussed in Section 6. The prior imposed upon γ_k is essentially a mixture model, with the mass dictated by θ. This then lets us compute the probability of the kth variable being globally active, which is

$$P(\gamma_k > 0 | \underline{y}) = P(b_k = 1 | \underline{y}) =: \hat{b}_k.$$

If $\hat{b}_k < g$ for some pre-determined $g \in (0, 1)$, then the variable is declared to be globally inactive. In GVS (and later SOLID) variables that are declared globally inactive are removed from the model and never considered again. After identifying globally active variables, GVS M posterior draws $\boldsymbol{\Omega}_1, \ldots, \boldsymbol{\Omega}_M$ (via Metropolis-Hastings within Gibbs), and calculates $\hat{f}_t = \hat{f}(\cdot | \boldsymbol{\Omega}_t)$ and posterior estimated maximums

$\hat{\boldsymbol{\chi}}_t = \arg\max \hat{f}_t$, and defines the estimated global maximizer to be

$$\hat{\boldsymbol{\chi}} = \arg\max_{\underline{x}} \left\{ \frac{1}{M} \sum_t \hat{f}_t \right\}.$$

GVS (and SOLID) have the same acquisition function, referred to as the *augmented expected improvement* (AEI) criterion, formally defined as

$$AEI(\underline{x}) = \mathbb{E}\left[\max\left\{\hat{f}(\underline{x}) - \hat{f}(\underline{x}^*), 0\right\}\right]\left(1 - \frac{\tau}{\sqrt{s^2(\underline{x}) + \tau^2}}\right),$$

where $\underline{x}^* = \arg\max_{\underline{x}_i \in \boldsymbol{X}} \left\{\hat{f}(\underline{x}_i) - \nu \cdot s(\underline{x}_i)\right\}$ for some $\nu \geq 0$, and τ is the nugget term. It's very similar to the ordinary EI criterion, not only in formula but in behavior. The difference, however, is that AEI will work in non-deterministic settings (noise is present), which is in fact the case here.

3 Bayesian Local Variable Selection

Identifying globally active/inactive variables is an excellent starting point for performing dimension reduction. However, global variable selection still may not be enough, as we still may be optimizing over a large number of globally active variables. In addition, implementing global variable selection requires a large number of iterations to get near the global maximum; in other words, global variable selection prioritizes exploration of the entire input space. Also, AEI can be hard to optimize over, since it balances exploration and exploitation. Therefore, we sacrifice a global maximum for a local maximum (and favor exploitation of our estimated maximum over exploration), which should take fewer runs to obtain. It therefore makes sense to optimize AEI in a neighborhood near $\hat{\boldsymbol{\chi}}$, since that is our current estimate of what we're trying to find. So, we're optimizing over some smaller region within the set of globally active variables, then some variables, even though globally active, may not contribute much to f in that region. Therefore, we introduce the notion of *locally active* and *locally inactive* variables. Simply put, a variable is *locally inactive* if it doesn't impact the values of f in a specified localized region.

We can determine if the kth variable is locally inactive by comparing the predictive surfaces with and without $\gamma_k \overset{\text{set}}{=} 0$. We denote the local space as $\mathcal{F}_t$, which centered at $\hat{\boldsymbol{\chi}}_t$ with size δ. We sample points in $\mathcal{F}_t$, and compare $\hat{f}_t$ with and without $\gamma_k \overset{\text{set}}{=} 0$. Figure 1 is a toy example that demonstrates the comparison we would make in a localized region, where $\hat{f}_k(x)$ sets $\gamma_k = 0$, and $\hat{f}(x)$ does not. For the plot on the left, note that $\hat{f}(x) \approx \hat{f}_k(x)$. This means that removing the kth variable didn't change the predictive surface much, and so we would conclude that the kth variable is locally inactive on the left plot. The plot on the right, on the other hand, show no meaningful relationship between $\hat{f}(x)$ and $\hat{f}_k(x)$. This means that removing the kth variable changed $\hat{f}$ a lot, and so we would declare the kth variable to be locally active.

A more precise algorithm is obtained by generating q prediction points $\boldsymbol{Q}_t$ centered near $\hat{\boldsymbol{\chi}}_t$, where

$$\boldsymbol{Q}_t \sim TN_{[0,1]^p}(\hat{\boldsymbol{\chi}}_t, \delta \boldsymbol{I}),$$

and δ controls how far away from $\hat{\boldsymbol{\chi}}_t$ we are allowing $\boldsymbol{Q}_t$ to go. At each $\boldsymbol{Q}_t$, we calculate $\hat{f}_t(\boldsymbol{Q}_t)$ and $\hat{f}_t^k(\boldsymbol{Q}_t)$, where $\hat{f}^k$ sets $\gamma_k = 0$. We then calculate

$$R_{kt}^2 = \text{Corr}\left(\hat{f}_t(\boldsymbol{Q}_t), \hat{f}_t^k(\boldsymbol{Q}_t)\right)^2,$$

where a larger R_{kt}^2 means that $\hat{f}_t(\boldsymbol{Q}_t) \approx \hat{f}_t^k(\boldsymbol{Q}_t)$. Lastly, we obtain $L_k = 1 - \frac{1}{M}\sum_{i=1}^{M} R_{ki}^2$. We subtract the value from 1 for ease of interpretability of L_k, so that larger values of L_k indicate a larger importance metric of the kth variable. Variable k is declared locally active if $L_k \geq \rho$ for some $\rho \in (0, 1)$. Algorithm 1 formally defines how to obtain locally active variables.

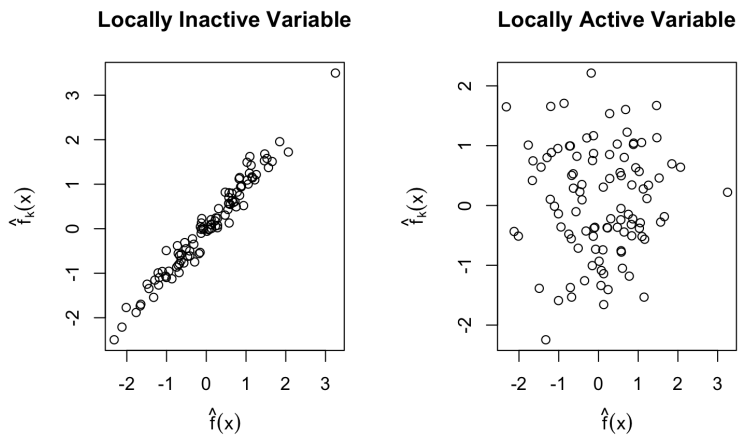

Figure 1: Demonstration of identifying a variable as locally active or inactive.

Algorithm 1 Identifying Locally Active Variables [3]

Initialize ρ and δ. Also randomly sample $m \leq M$ posterior draws.
for $t \in \{1, \ldots, m\}$ **do**
    Estimate $\hat{\boldsymbol{\chi}}_t$ using $\boldsymbol{\Omega}_t$ and all globally active variables.
    Construct q prediction points $\boldsymbol{Q}_t$ centered around $\boldsymbol{\chi}_t$.
    Determine baseline predictions $\hat{f}$ at $\boldsymbol{Q}_t$ using $\boldsymbol{\Omega}_t$.
    for variable $k \in \{1, \ldots, p\}$ **do**
        Make alternative predictions $\hat{f}_t^k$ at $\boldsymbol{Q}_t$ and calculate R_{kt}^2.
    end for
end for
Calculate $L_k = 1 - \bar{R}_{k\cdot}^2$.
return $\boldsymbol{A} = \{k : L_k \geq \rho | \hat{\boldsymbol{\chi}}\}$, the set of locally active variables.

4 Sequential Optimization using SOLID

Now, it's time to finally introduce the novel algorithm in the paper, SOLID. The precise algorithm of SOLID is give in Algorithm 2. I've previously dropped some hints about what SOLID does, but the general idea is that SOLID performs global and local variable selection at each iteration. In other words, it's an extension of GVS that includes local variable selection.

It is imperative to point out what is happening in the optimization stage. While SOLID will favor exploitation over exploration, it still considers some exploration. So, after we've obtained the set of locally active variables $\boldsymbol{A}$, we consider two regions to optimize over. For each locally active variable, define

$$\mathcal{R}^{\delta} = [\min(\hat{\chi}_{k,1}, \ldots, \hat{\chi}_{k,m}) - \delta, \max(\hat{\chi}_{k,1}, \ldots, \hat{\chi}_{k,m}) + \delta] \text{ and } \mathcal{R}^{\boldsymbol{A}} = [0, 1].$$

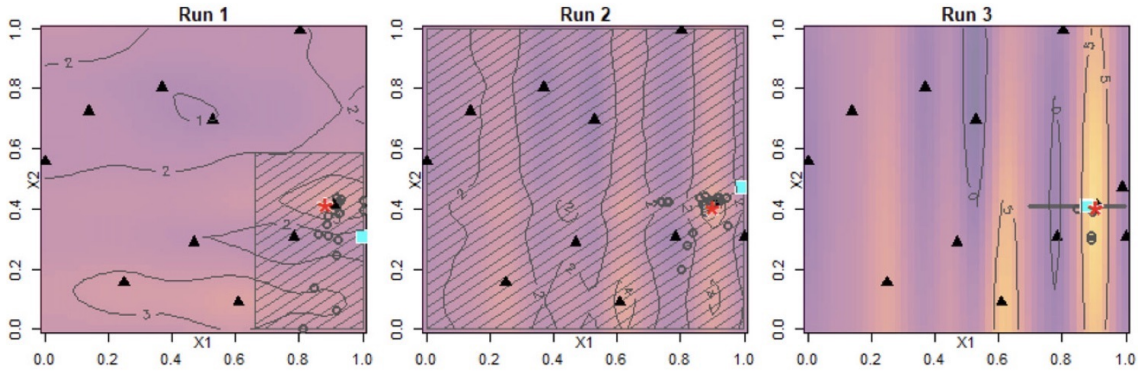

Figure 2: Several iterations of a particular implementation of SOLID on a 2d function [3].

This means that SOLID redefines AEI with respect to $\boldsymbol{A}$. The notion of $\mathcal{R}^{\delta}$ is very similar to what has been discussed in Response Surface Methodology (RSM), where we are performing a local neighborhood search of our function. At each iteration in SOLID, AEI is evaluated at both $\mathcal{R}^{\delta}$ and $\mathcal{R}^{\boldsymbol{A}}$, where we choose the largest AEI of the two. This balances exploration (via $\mathcal{R}^{\boldsymbol{A}}$) and exploitation ($\mathcal{R}^{\delta}$), where we want to balance our current belief as to where $\boldsymbol{\chi}$ is, but also acknowledge that we may be approaching a local maximum. Once the acquisition function gives us a new point to sample at, we sample it, and repeat until some stopping criterion is reached. Since locally inactive variables are set to their values in $\hat{\boldsymbol{\chi}}$, we are still exploiting our knowledge about $\hat{\boldsymbol{\chi}}$. In other words, SOLID exploits the locally active dimensions.

Note that, while variables declared to be globally inactive are permanently removed from the model, variables declared to be locally inactive are re-introduced. This is to erase any misclassifications made in the earlier step.

Shown in Figure 2 is a walkthrough of how SOLID works. In the first iteration (Run 1), we are presented with the initial design points (▲), our estimated global maximum (∗), and the predicted surface $\hat{f}$ (shading). For this iteration, we chose $\mathcal{R}^{\delta}$, since the shaded rectangle doesn't cover the entire locally active variable space. Our posterior draws $\hat{\boldsymbol{\chi}}_t$ (∘) are also displayed, as is the AEI maximizer (from $\mathcal{R}^{\delta}$ this time) (■). We then sample at the AEI maximizer (the blue square becomes a triangle), and move on to the second iteration (Run 2). This time, the AEI maximizer was in $\mathcal{R}^{\boldsymbol{A}}$, since we are sampling from the entire regions of $X1$ and $X2$. In Run 3, we determined that $X2$ was locally inactive (as evidenced by the fact that we don't vary $X2$ in $\mathcal{R}^{\delta}$). SOLID would then continue to execute until

some stopping criterion is reached.

Algorithm 2 SOLID [3]

Set n_0 and N (the maximum number of evaluations). Also set g, δ, ρ, M, m, and c.
Create an initial maximin LHS(n_0, p) design $\boldsymbol{X}$.
Generate $\underline{y}$ from $Y(\boldsymbol{X})$.
for $i \in \{0, \ldots, N\}$ **do**
    Obtain M posterior draws of $\boldsymbol{\Omega}_t$ and $\boldsymbol{\chi}_t$, and calculate $\hat{f}$ and $\hat{\boldsymbol{\chi}}$.
    Global Variable Selection: Remove variables with $\hat{b}_k < g$ from $\boldsymbol{X}$. If variables are removed, repeat the previous step with the new $\boldsymbol{X}$.
    Local Variable Selection: Obtain $\boldsymbol{A}$ form previous algorithm.
    Define restricted $\mathcal{R}^{\delta}$ and unrestricted $\mathcal{R}^{\boldsymbol{A}}$ search spaces.
    Localized Optimum Estimation: Update estimate $\hat{\boldsymbol{\chi}}$ in $\mathcal{R}^{\boldsymbol{A}}$ using $\hat{f}$. Store as $\hat{\boldsymbol{\chi}}^i$.
    Create maximin LHS designs $\boldsymbol{C}_{\delta} \subset \mathcal{R}^{\delta}$ and $\boldsymbol{C}_{\boldsymbol{A}} \subset \mathcal{R}^{\boldsymbol{A}}$.
    Evaluate AEI in $\boldsymbol{C}_{\delta}$ and $\boldsymbol{C}_{\boldsymbol{A}}$. Define the set with the largest AEI as $\boldsymbol{C}$.
    Localized AEI Estimation: Perform line search optimization to identify $\underline{x}^* = \arg\max_{\underline{x} \in \boldsymbol{C}} AEI(\underline{x})$.
    Augment $\underline{x}^*$ to $\boldsymbol{X}$. Generate $Y(\underline{x}^*)$ and add to $\underline{y}$.
end for
return $\{\hat{\boldsymbol{\chi}}^0, \ldots, \hat{\boldsymbol{\chi}}^N\}$.

5 Simulation Study and Application to Sarcos Robot Data

The paper investigates the overall efficiency of SOLID compared to three other sequential optimization methods: GVS, None (does not perform any global variable selection), and Oracle (performs optimization only on the globally active variables). Note that Oracle knows which variables are globally active, so ideally SOLID would perform similarly to Oracle in terms of mean improvement.

The metric we will use to compare these four methods is *mean improvement*, defined as $\frac{1}{n}\sum_{i=1}^{n} f(\hat{\boldsymbol{\chi}}_n) - f(\hat{\boldsymbol{\chi}}_0)$, which measures how much our expected maximum increases from our initial guess. While most literature would likely use *regret* to measure optimizer performance, we acknowledge that we are encouraging SOLID to find local maximums in an attempt to reduce the number of necessary runs. Due to this, regret would not be an ideal candidate.

Shown in Figure 3 is the mean improvement of the methods across three functions that are hard to optimize: Beach, Drum, and Simba. As expected, Oracle had the best mean improvement for Beach and Drum (since it knows which variables to consider), and None did the worst, since it is trying to optimize with nuisance variables. SOLID was initially the second-best option for mean improvement,

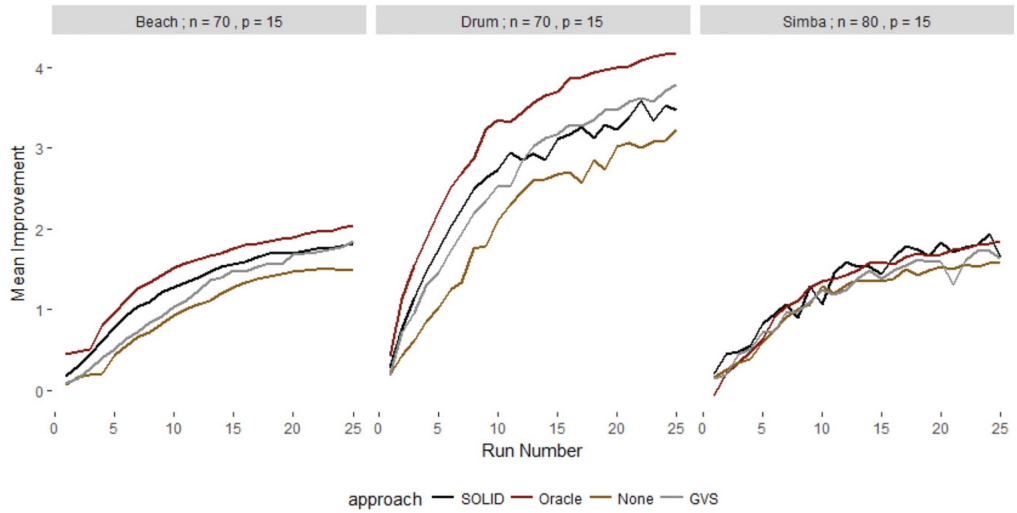

Figure 3: Mean Improvement of the four candidate optimization methods across the Beach, Drum, and Simba functions.

but then GVS (the third best) catches up with it. This is a good result, since SOLID sought to optimize f in a fewer number of iterations, which we can see is, in fact, happening. The same trend is present in Simba, but none of the algorithms improved that quickly, so any differences might not be important.

While SOLID might require fewer iterations to optimize f than GVS and None, an important point is that SOLID was by far the slowest of the four algorithms, which may be due to the local variable selection. Per hour that Oracle took to add 25 design points, None took 1.4 hours, GVS took 4, and SOLID took 5.3. So the benefit from fewer iterations might be (at least partially) canceled out by other methods need less time per point.

Next, SOLID is applied to the Sarcos robot dataset [4], where a robot arm is drawing a figure eight. The input variables include the positions, velocities, and accelerations of seven different points on a robot as it draws. The response variable is the first of seven joint torque measurements.

As was suggested in the simulation, SOLID had much better mean improvement over GVS, which could be due to the fact that SOLID used significantly fewer variables at each iteration than GVS (15.88 variables for SOLID, versus 21 for GVS). Another note is that neither SOLID nor GVS found any variables to be globally inactive, so GVS would actually be worse than None in this case, since GVS would waste additional resources trying to predict if a variable is globally inactive, but ultimately end up optimizing over all of the variables anyways.

6 Discussion

SOLID is a sequential algorithm, designed for finding the maximum value of some function in a computer experiments setting, that attempts to speed up the optimization process by performing global and local variable selection, and optimizing over a reduced input space. The simulation studies confirmed that SOLID did take fewer iterations to reach the same conclusion that competing methods require more runs for, indicating some usage in terms of expected improvement.

One weakness of SOLID was actually mentioned in the paper itself. The global variable selection algorithm in SOLID is relatively slow, given the same number of runs for other methods, which could be due to the fact that MCMC that is required to obtain these posterior estimates. Other methods, such as random embedding [6], might be faster. A modification to SOLID that would speed up global variable selection would make SOLID much more practical than other methods.

Another area of improvement is the choices of cutoff variables for both global and local variable selection, which are g and ρ, respectively. These values were conventionally chosen, but an adaptive method would improve this process. Another idea would be to investigate which fixed cutoff values would be ideal.

Next, the authors included some odd proposal distributions. When referring to global variable selection. Recall that, if $b_k = 0$, then u_k is updated via a sliding uniform candidate distribution. More specifically,

$$u_k \sim \text{Unif}\left(\max\{0, u_k - 50\epsilon(u_k)\}, u_k + \epsilon(u_k)\right), \text{ where } \epsilon(u_k) = \begin{cases} \min\{50, u_k h\} & , u_k \geq 30 \\ \max\{1, u_k h\} & , u_k \in [0, 30) \end{cases},$$

and $h \sim \text{Unif}\left(\frac{1}{2}, 2\right)$. This proposal indirectly penalizes u_k towards smaller values. However, if a penalty is desired, it really should be in the prior.

The paper considers a space-filling maximin LHS as an initial design, since this design should be better at identifying potential regions where $\boldsymbol{\chi}$ may be. However, I would instead propose a cascading

LHS design. It has been shown that space-filling designs are not ideal for estimating hyperparameters [7]. It will be demonstrated later how important it will be to estimate γ_k well, and cascading LHS designs are less (but still) space-filling by construction. However, the justification provided in [7] is only supported by minimal evidence, and could be a plan of future work.

In addition, there exists the choice in GVS and SOLID to never re-introduce variables that are declared to be globally inactive. I do not believe that this a good idea, because we are estimating γ_k. Comparing $\hat{b}_k$ to g is essentially like rejecting the null hypothesis when the p-value$< \alpha$ in elementary statistical inference. However, we always run the chance of committing a Type I Error. In a sequential algorithm, we should consider the possibility of incorrectly removing a variable, and introduce a method to re-introduce variables that were removed earlier.

Finally, there are several tuning parameters needed in SOLID. I've already touched upon g and ρ for global and local variable selection, respectively, but there is also δ used to generate $\boldsymbol{Q}_t$. All of these tuning parameters can change how SOLID behaves greatly, and more deterministic methods/formulas can shrink subjective impact on SOLID.

References

[1] Matthias Schonlau. *Computer experiments and global optimization*. PhD thesis, University of Waterloo, 1997.

[2] Robert B Gramacy. *Surrogates: Gaussian process modeling, design, and optimization for the applied sciences*. Chapman and Hall/CRC, 2020.

[3] Munir A. Winkel, Jonathan W. Stallrich, Curtis B. Storlie, and Brian J. Reich and. Sequential optimization in locally important dimensions. *Technometrics*, 63(2):236–248, 2021.

[4] Sethu Vijayakumar and Stefan Schaal. Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space. In *Proceedings of the seventeenth international conference on machine learning (ICML 2000)*, volume 1, pages 288–293. Morgan Kaufmann Burlington, MA, 2000.

[5] Crystal Linkletter, Derek Bingham, Nicholas Hengartner, David Higdon, and Kenny Q Ye. Variable selection for gaussian process models in computer experiments. *Technometrics*, 48(4):478–490, 2006.

[6] Nando de Freitas and Ziyu Wang. Bayesian optimization in high dimensions via random embeddings. 2013.

[7] Boya Zhang, D Austin Cole, and Robert B Gramacy. Distance-distributed design for gaussian process surrogates. *Technometrics*, 63(1):40–52, 2021.