

Supplementary Materials

1 Details on SOLID Algorithm

1.1 Maximizing AEI Using Local Information

Let \mathbb{A} denote the estimated set of locally active variables at $\hat{\chi}$ for the current design. Likewise, let $\mathbb{A}_{\hat{\chi}}$ denote the active subspace of $[0, 1]^p$ where inactive variables are projected to their optimal inputs in $\hat{\chi}$. Correctly specifying $\mathbb{A}_{\hat{\chi}}$ can greatly reduce the difficulty of optimizing the *AEI* surface by focusing on a subspace exhibiting substantial variation of the response surface near the estimated global maximizer. We next detail a strategy for maximizing *AEI* using this local importance information.

It is nontrivial to find the optimal input for *AEI* since the *AEI* surface could be multi-modal. We simplify the optimization by reducing the dimension to $\mathbb{A}_{\hat{\chi}}$, fixing all inactive variables to their values estimated by $\hat{\chi}$. Even still, this space may still be large, and there is the question of whether we should explore the entire subspace or just a neighborhood of the subspace centered around $\hat{\chi}$. After all, if $\hat{\chi}$ is truly near the global maximizer, restricting the search for the *AEI* maximizer to a neighborhood of $\hat{\chi}$ could be advantageous.

First, we evaluate *AEI* over C candidate points that form a maximin LHS design (Kleijnen, 2015) in $\mathbb{A}_{\hat{\chi}}$. Denote this candidate set by $\mathbf{C}_{\mathbb{A}}$. We then construct another set of C candidate points, denoted $\mathbf{C}_{\mathbb{A}\delta}$, that form a maximin LHS design in a δ -neighborhood of $\hat{\chi}$ contained in $\mathbb{A}_{\hat{\chi}}$. To create this neighborhood, for each $k \in \mathbb{A}$, we set the restricted lower and upper limits to be $\max\{0, \min(\hat{\chi}_{k1}, \dots, \hat{\chi}_{km}) - \delta\}$ and $\min\{1, \max(\hat{\chi}_{k1}, \dots, \hat{\chi}_{km}) + \delta\}$ using the m posterior draws of each estimate of the k -th coordinate of the global maximizer, denoted $\hat{\chi}_{kt}$ for the t -th draw. SOLID evaluates *AEI* at the $2C$ candidate points in $\mathbf{C}_{\mathbb{A}\delta}$ and $\mathbf{C}_{\mathbb{A}}$. Guidance for choosing δ is given in Section 1.2.

At this stage, it is possible to combine both sets of candidate points into one large set, \mathbf{C} . In practice, we find that the candidate points with the greatest *AEI* are often all in one of the two sets. In our implementation, whichever set has the largest *AEI* becomes the set of candidate points \mathbf{C} for the next step. Conceptually, this helps us see if SOLID is honing-in on a restricted space $\mathbf{C}_{\mathbb{A}\delta}$ or exploring the larger space $\mathbf{C}_{\mathbb{A}}$.

Next, we choose the five candidate points with the greatest *AEI* and perform local line searches in the direction of the $|\mathbb{A}|$ -dimensional *AEI* gradient using only the locally active dimensions. The *AEI* gradients of the locally inactive variables are set to zero. We place two restrictions on the local line search. First, we require that the line search lie within a p -dimensional ball of radius δ centered at \mathbf{x}^* , the starting point. Second, whenever the line search proposes a coordinate outside the design space, that coordinate is set to equal the boundary value and the corresponding coordinate of the gradient is set to zero. This approach was inspired by the more rigorous approach of Rosen (1960). Once the five line searches are complete, the one with the largest *AEI* is chosen as the next design point.

1.2 Choosing the δ Tuning Parameter

The δ term handles several aspects of SOLID's performance. First, it controls the proportion of the design space that is used to establish local importance. Second, δ determines how far away the restricted candidate points $\mathbf{C}_{\mathbb{A}\delta}$ can be from the estimated maximizer $\hat{\chi}$. Third, it limits how far the local line searches for *AEI* can go. Simulation studies show that local importance is fairly constant for $\delta > 0.50$. In choosing a δ value, an important requirement is that the mean of the prediction points (whose distance from the global maximizer depends on δ , decreases as δ increases). As an alternative to specifying δ directly, one could choose δ such that the mean of the prediction points is, say, 80% of the estimated global maximum.

Algorithm 1 SOLID: Sequential Optimization in Locally Important Dimensions

```

1: procedure SEQUENTIAL DESIGN
2:   Create an initial maximin LHS( $n, p$ ) design,  $\mathbf{X}$ 
3:   Evaluate  $Y(\mathbf{X}) = f(\mathbf{X}) + \epsilon$ 
4:   Fit a Gaussian process and obtain  $\boldsymbol{\Omega}_1, \dots, \boldsymbol{\Omega}_M$  draws from the joint posterior distribution
5:   for step  $i \in \{1, \dots, N\}$  do
6:     Perform global variable selection, GVS( $\boldsymbol{\Omega}, \kappa$ )
7:     Update parameter estimates  $\boldsymbol{\Omega}_1, \dots, \boldsymbol{\Omega}_M$  after global variable selection
8:     Estimate set of locally active variables  $\mathbb{A}$  using LVS( $\boldsymbol{\Omega}, i$ )
9:     Construct a matrix of candidate points  $\mathbf{C}$  using CANDIDATES( $\mathbb{A}, \boldsymbol{\Omega}, \delta$ )
10:    Choose the design point  $\mathbf{x}^{**}$  that maximizes  $AEI$  using AEI( $\mathbf{C}, \bar{\boldsymbol{\Omega}}$ )
11:    Augment  $\mathbf{x}^*$  to  $\mathbf{X}$  and  $Y(\mathbf{x}^*)$  to  $Y$ 
12:    Update parameter estimates again
13:    Estimate the optimal input  $\hat{\chi}$ .

14: function GVS( $\boldsymbol{\Omega}, \kappa$ )
15:   for variable  $k \in \{1, \dots, p\}$  do
16:     Drop  $X_k$  from the design matrix  $\iff Pr(\gamma_k > 0 \mid \mathbf{y}, \bar{\boldsymbol{\Omega}}) < \kappa$ 

17: function LVS( $\boldsymbol{\Omega}, i$ )
18:   Randomly sample  $m < M$  posterior draws of the optimal design point  $\hat{\chi}_1, \dots, \hat{\chi}_m$ 
19:   for  $t \in \{1, \dots, m\}$  do
20:     Make predictions  $\hat{f} \mid \boldsymbol{\Omega}_t$  at  $q$  points using  $\mathbf{X}$ 
21:     for variable  $k \in \{1, \dots, p\}$  do
22:       Set  $\gamma_k = 0$  and make alternative predictions,  $\hat{f}^k$  at the same  $q$  points
23:       Calculate the local importance,  $L_{tk} \equiv 1 - \text{Corr}(\hat{f}, \hat{f}^k \mid \boldsymbol{\Omega}_t)^2$ 
24:   Summarize across posterior draws and calculate  $L_k = \text{mean}(L_{1k}, \dots, L_{mk})$  for  $k \in \{1, \dots, p\}$ .
25:   Let  $\mathbb{A}_i = \{k : L_k \geq \rho \mid \rho \in (0, 1)\}$  be the set of locally active variables at step  $i$ . return  $\mathbb{A}_i$ 

26: function CANDIDATES( $\mathbb{A}, \boldsymbol{\Omega}, \delta$ )
27:   Create  $\mathbf{C}_{\mathbb{A}\delta}$  and fill the locally active dimensions with a maximin LHS( $C, |\mathbb{A}|$ ) design,
   rescaled such that the bounds are restricted to be between  $\max\{0, \min(\hat{\chi}_{k1}, \dots, \hat{\chi}_{km}) - \delta\}$  and
    $\min\{1, \max(\hat{\chi}_{k1}, \dots, \hat{\chi}_{km}) + \delta\}$  for  $k \in \mathbb{A}$ , using the estimated optimal inputs  $\hat{\chi}_\ell$  for  $\ell \in \{1, \dots, m\}$ 
28:   Fix the remaining dimensions at the estimated optimal input  $\hat{\chi}_k$  for  $k \notin \mathbb{A}$ 
29:   Create  $\mathbf{C}_{\mathbb{A}}$ , a maximin LHS( $C, p$ ) design  $\in [0, 1]^p$ , and calculate the largest  $AEI$  on  $\mathbf{C}_{\mathbb{A}}$ .
30:   if  $\max_{\mathbf{x} \in \mathbf{C}_{\mathbb{A}}} AEI(\mathbf{x}) > \max_{\mathbf{x} \in \mathbf{C}_{\mathbb{A}\delta}} AEI(\mathbf{x})$  then  $\mathbf{C} \equiv \mathbf{C}_{\mathbb{A}}$ 
31:   else  $\mathbf{C} \equiv \mathbf{C}_{\mathbb{A}\delta}$  return  $\mathbf{C}$ 

32: function AEI( $\mathbb{A}, \mathbf{C}, \bar{\boldsymbol{\Omega}}$ )
33:   Evaluate  $AEI$  at each of the candidate points  $\mathbf{C}$ 
34:   Set  $\mathbf{x}^{(i)} = \arg \max_{\mathbf{x} \in \mathbf{C}} AEI(\mathbf{x})$ 
35:   if  $k \in \mathbb{A}$  then compute  $g_k$ , the  $k^{th}$  component of the gradient of  $AEI(\mathbf{x}^*)$ 
36:   else  $g_k = 0$ 
37:   Choose the 5 design points with the largest  $AEI$ 
38:   For each of the 5 design points (WLOG:  $\mathbf{x}^{(i)}$ ), do five line searches spanned by  $\mathbf{x}^{(i)} + t\mathbf{g}$  for different
   step multipliers,  $t \in \left[0, \delta(g_1^2 + \dots + g_p^2)^{-1/2}\right]$ , where the bounds keep the line searches within a radius
    $\delta$  of the starting point, and each subsequent line search begins where the previous one ends. return
 $\mathbf{x}^{**} \equiv \arg \max_t AEI(\mathbf{x}^{(i)} + t\mathbf{g})$ 

```

2 Results for Additional Simulation Test Functions

This section includes results for three additional test functions that may be found under the Optimization link at <https://www.sfu.ca/ssurjano/index.html>. We refer to these functions as Ackley, Michalewicz, and Hartmann. The first six variables are globally active in all cases and we begin with a candidate set of 15 variables. We explore the behavior of SOLID, GVS, Oracle, and None with 10 additional runs. The other settings of the simulation follow those described in Section 5 of the main paper; the initial design size was fixed at $n_0 = 70$. The Ackley function has many local maxima around the border of the design space, and a large peak in the center of the design space. The Michalewicz function has many valleys and ridges, with much of the design space having a flat floor. There are many local optima but they can be difficult to find with a small initial design. The 6 dimensional Hartmann function is not as straightforward to describe as these other functions, but it does have multiple maxima.

Figure 1 shows the mean improvement of all methods over the true value of the initial estimated maximum, $\hat{\chi}^0$, based on the None method. The Ackley function initially showed little differentiation between the non-Oracle methods, but runs 7 to 10 began to show an improvement of SOLID and GVS over None. Indeed, it was not until the later runs that SOLID and GVS began removing variables. The Michalewicz function gave very unusual results. Oracle performed worse than the other methods, with a negative mean improvement across all runs. None performed best with SOLID closely behind. We attribute these results to the flat floor behavior noted in the previous paragraph that led to poor estimates of the γ_k . A larger n_0 would likely improve these results. Finally, the Hartmann function showed clear improvement of the variable selection methods over the None approach. SOLID differentiated from GVS the most in run number 1. As will be shown later, nearly all 6 globally important variables were also deemed locally important which explains the lack of separation between SOLID and GVS.

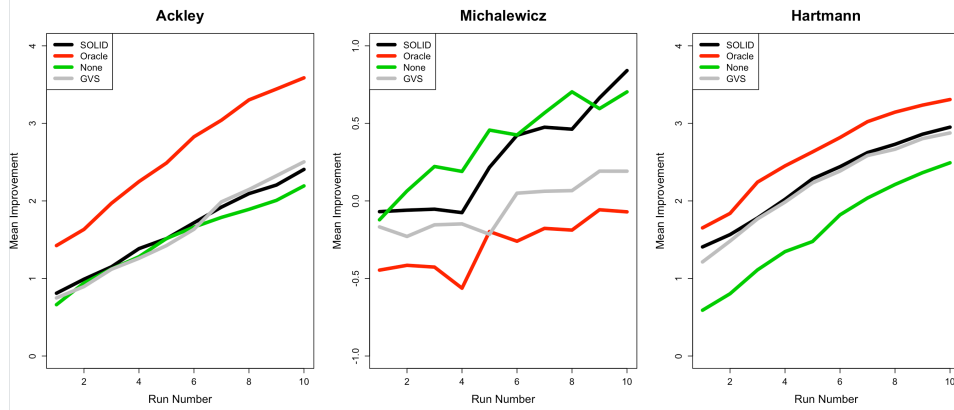


Figure 1: Mean relative improvement ($f(\hat{\chi}^i) - f(\hat{\chi}^0)$) across 100 simulations, using the three test functions for sequential runs $i \in \{1, \dots, 10\}$. Improvement at run 0 is not shown as all approaches have zero relative improvement.

Table 1 shows the mean overall improvement for the three test functions across the 10 added runs. No significant differences were found between GVS and SOLID under the Wilcoxon rank sums test, but GVS and SOLID were both significantly better than None for the Hartmann function ($p < 0.001$). There were no significant differences between the non-Oracle approaches for the Ackley. Both None and SOLID were significantly different from Oracle for the Michalewicz function ($p < 0.001$ and $p = 0.002$).

Next, we investigated the local importance performance of SOLID for these test functions. Figure 2 shows boxplots of the L_k across all 100 simulated data sets and 10 runs. Unlike the test functions in the main paper, the median L_k values rarely exceeded 0.2 even for the globally active variables. This suggests that the SOLID algorithm settings may not be ideal for these functions. The initial sample size may have been too small or the local importance parameter δ was too large. For Ackley, we clearly see that the globally active variables were often deemed locally active. However, for Michalewicz, all variables had similar L_k distributions. For Hartmann we can see that variable 3 was often deemed locally unimportant while the remaining globally active variables tended to be locally important. This explains the slight improvement of SOLID over GVS.

Table 1: The mean improvement for all approaches and test functions averaging across 100 simulated initial designs and 10 runs. A Wilcoxon rank sums test shows that SOLID and GVS performs similarly across all functions.

Test Function	Oracle		SOLID		GVS		None		GVS \neq SOLID P-value
	Mean	Std err	Mean	Std err	Mean	Std err	Mean	Std err	
Ackley	2.60	0.106	1.62	0.107	1.60	0.102	1.51	0.088	0.781
Michalewicz	-0.28	0.114	0.28	0.103	-0.03	0.105	0.38	0.086	0.181
Hartmann	2.63	0.140	2.27	0.137	2.20	0.126	1.63	0.103	0.690

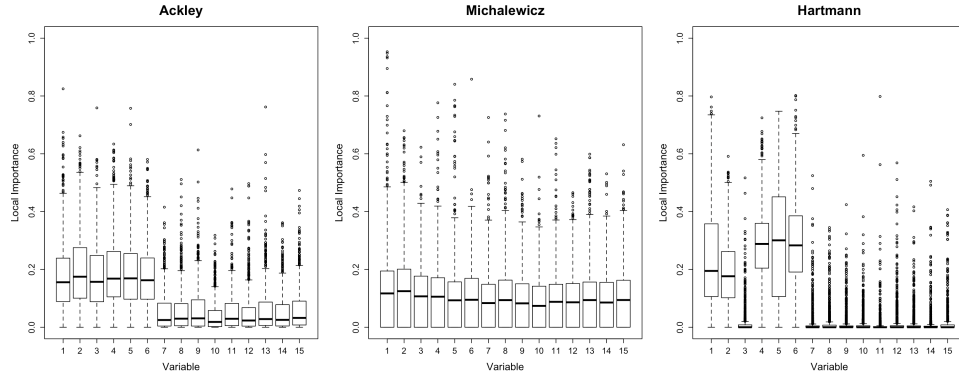


Figure 2: L_k boxplots over all 100 simulated datasets and 10 runs. The first six boxplots are for the globally active variables.

Table 2 summarizes the variable selection performance of the four approaches after run 10. Both GVS and SOLID were unable to remove most of the unimportant variables for the Ackley and Michalewicz, but performed quite well for the Hartmann function. For Ackley, SOLID used 9.73 variables on average for optimization, while GVS used 11.96. For Hartmann, SOLID used 5.25 variables on average while GVS used 6.93; the former was due to designating variable 3 as locally inactive. These functions were clearly challenging and point out areas for improvement for the SOLID algorithm, such as choosing the appropriate initial run size and SOLID threshold parameters g , ρ , and δ .

Table 2: The false positives are variables that are included in the design but are truly globally inactive. There are 15 total variables; at most 9 are false positives.

Mean Number of Variables at run 10 (across 100 simulations)

Test Function Approach	Used for Optimization			False Positives		
	Ackley	Michalewicz	Hartmann	Ackley	Michalewicz	Hartmann
Oracle	6.00	6.00	6.00	0.00	0.00	0.00
SOLID	9.73	9.31	5.25	7.46	5.93	2.73
GVS	11.96	11.73	6.93	6.04	6.92	1.67
None	15.00	15.00	15.00	9.00	9.00	9.00

3 R Code for Test Functions Drum, Beach, and Simba

```
truth <- function( X , P, name = 'simba'){
  # P should be the number of globally active variables
  # for the test function to work.
  # This section of the code keeps track of which
  # variables are actually used. If X3 is required, but
  # X does not have a column called "X3," then the "truth"
  # function will input a random number for that variable.
  if( !is.null( colnames(X) ) ){

    neednames = paste("X",1:P, sep = "")
    xnames = colnames(X)

    ## unless variables given, we use random uniform entries
    XX = matrix( runif(length(X)*P), nrow(X), P)

    XX[, is.element(neednames, xnames) ] = X
    X <- XX
  } else # if we know the column names, use them

    if( is.null(colnames(X)) & ncol(X) < P ){

      need = P - ncol(X)
      XX = matrix( runif(length(X)*P), nrow(X), P)

      colnames(XX) = paste("X", 1:P , sep="")
      XX[, 1:ncol(X)] = X
      X <- XX
    }# otherwise, we just require dimensions to match
  if(name == 'toy'){

    x1 = X[,1] ; x2 = X[,2];

    Y = 10*(x2)^2*pnorm( 10*(x1 - .4 ) ) +
      sin(5*pi*( x2 - x1^2 ) - x1*x2 )*pnorm( 10*(.4 - x1 ) )

    return(Y)
  } else # end toy

  if(name == 'drum'){

    x1 = X[,1] ; x2 = X[,2]; x3 = X[,3];
    x4 = X[,4] ; x5 = X[,5]; x6 = X[,6]

    inner = (6 - x3/4)*cos( 4*pi*( x3-.5) )*(
      dnorm( 11*( (x1 -.5)^2 + (x2-.5)^2 + .25 )^2 )
    ) * (1 - 3*(x1-.3)^2)

    middl = (1 + 2*x4)*sin( 2*pi* x4*( x5-.3) )*(
```

```

    pnorm(      6*( (x1 -.5)^2 + (x2-.5)^2 - 0.13 ))
    *pnorm(     -8*( (x1 -.5)^2 + (x2-.5)^2 - 0.11 ))
  )*(
    (      1 + x4^2 + x5^2*(x2-.2)      )

outer =      (1 - 2*x5)*cos(  2*pi*(x5) * (x4+.5)      )*(
    pnorm(      8*( (x1 -.5)^2 + (x2-.5)^2 - 0.2 )      )
  )*(
    (      1 - x6^2 - x5^2*(x1+.2)      )

Y = (inner + middl + outer) *(10/2.032078) - 4.4831

return(Y)

} else # end drum

if(name == 'beach'){

  x1 = X[,1] ; x2 = X[,2]; x3 = X[,3];
  x4 = X[,4] ; x5 = X[,5]; x6 = X[,6]

  sdz = .2 + (2 + x6 + x5 - 1.5*x1)*(3 + x4 - x3 - x2)/12

  bumps = (

    5*sin(6*pi*x1*x6)*(x3^2 +1)-( x2^2 + 4 - x1*x2/(x3 - 7) + x4*(x5 - .3))^2*
      cos( 4*pi*x1*x3^2 )^10*(x1*x2^2 - .5)*(x2*x6 - .5)*(x5 - .5)

  )*(
    ( pnorm( 10*( .8 - x3 ), sd = sdz )
      *pnorm( 1*( x1 - .1 ), sd = sdz )
      *pnorm( 1*( x2 - .1 ), sd = 2*sdz )
      # *pnorm( 1*( x3 - .1 ), sd = 2*sdz )
    )

  horiz = ( 10.5 - 30*(x1 - .3)^2 )*(
    ( pnorm( 1*( .2 - x3 ), sd = sdz )
      *pnorm( 10*( .3 - x2 ), sd = sdz ) )

  vert = ( 10.5 - 30*(x2 - .85)^2 )*(
    ( pnorm( 5*( x3 - .8 ), sd = sdz )
      *pnorm( 10*( x1 - .8 ), sd = sdz ) )

  Y = bumps + horiz + vert -.97013 + .470418

  return(Y)

} else # end beach

if(name == 'simba'){

```

```

x1 = X[,1] ; x2 = X[,2]; x3 = X[,3];
x4 = X[,4] ; x5 = X[,5]; x6 = X[,6]

Y = 3.14749 + sin( 2*pi*(x1^2 - 2*x2*(1 + x3) ) ) *
  ( pnorm( 30 *(x2 -.3)) + pnorm(30*(.8 - x2)) - 1 ) *
  2*sin( 4*pi*x1 + 3*pi*(1+x3) + 2*pi*(x4 + x5) + 3*pi*(1 + x6)) +

  ( 4 + 6*(x1) ) *
  (( pnorm( 30*(x2 - .0)) + pnorm(30*(.2 - x2)) ) - 1 ) *
  (( pnorm( 30*(x1 - .0)) + pnorm(30*(.6 - x1)) ) - 1 ) *
  ( pnorm( 10*(.2 - x3)) ) +

  ( 1 - 8*(x1 + x2 - x4 - x5 - x6)^2 ) *
  (( pnorm( 40*(x2 - .0)) + pnorm(40*(.2 - x2)) ) - 1 ) *
  (( pnorm( 40*(x1 - .6)) + pnorm(40*(1 - x1)) ) - 1 ) *
  ( pnorm( 10*(.2 - x3)) ) +

  .5*( 1 - sin( 8*pi*x1 + 7*pi*x2*x3 - 4*pi*x4*x5*x6) ) *
  (( pnorm( 30*(x2 - .0)) + pnorm(30*(.3 - x2)) ) - 1 ) *
  ( pnorm( 8*(x3 - .3)) ) +

  ( 5*cos( 2*(x2 + .5)*(-x4 + .5)*(-x5+.5)^2 )*(-x6 - .5) -
    .02*((1-x2)^2 +
      (1-x1)^2 +
      (1-x3 - .3*x4)^2 +
      (1-x5 + .5*x4)^2 +
      (.8-x6 - .4*x4)^2 ) ) *
  ( pnorm(5*(x2 - 1) + pnorm(10*(.5 - x3))))

return(Y)

}# end simba
}# end function

# maximum for Simba
X = matrix( c(.523, .0999, 0,0.298, 0.298, 0.245), ncol= 6)
truth(X, name = "simba", P = 6)

# maximum for Beach
X = matrix( c( 1, 0.85, 1, 0, 0,0), ncol= 6)
truth(X, name = "beach", P = 6)

# maximum for Drum
X = matrix( c( .368, .533, 0, 1, 0.555, 1), ncol= 6)
truth(X, name = "drum", P = 6)

```

References

Hastings, W. K. (1970), "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, 57, 97–109.

Kleijnen, J. P. C. (2015), *Design and Analysis of Simulation Experiments*, Springer Publishing Company, Incorporated, 2nd ed.

Rosen, J. B. (1960), "The Gradient Projection Method for Nonlinear Programming. Part I. Linear Constraints," 8, 181–217.