Algorithm 1028: VTMOP: Solver for Blackbox Multiobjective Optimization Problems

TYLER H. CHANG, Virginia Polytechnic Institute and State University, Dept. of Computer Science and Argonne National Laboratory, Mathematics and Computer Science Division
LAYNE T. WATSON, Virginia Polytechnic Institute and State University, Depts. of Computer Science, Mathematics, and Aerospace and Ocean Engineering
JEFFREY LARSON, Argonne National Laboratory, Mathematics and Computer Science Division
NICOLE NEVEU, SLAC National Accelerator Laboratory
WILLIAM I. THACKER, Winthrop University
SHUBHANGI DESHPANDE, Oracle Labs
THOMAS C. H. LUX, Virginia Polytechnic Institute and State University, Dept. of Computer Science

VTMOP is a Fortran 2008 software package containing two Fortran modules for solving computationally expensive bound-constrained blackbox multiobjective optimization problems. VTMOP implements the algorithm of [32], which handles two or more objectives, does not require any derivatives, and produces well-distributed points over the Pareto front. The first module contains a general framework for solving multiobjective optimization problems by combining response surface methodology, trust region methodology, and an adaptive weighting scheme. The second module features a driver subroutine that implements this framework when the objective functions can be wrapped as a Fortran subroutine. Support is provided for both serial and parallel execution paradigms, and VTMOP is demonstrated on several test problems as well as one real-world problem in the area of particle accelerator optimization.

CCS Concepts: • **Mathematics of computing → Mathematical optimization**; • **Applied computing → Multi-criterion optimization and decision-making;**

This work was supported in part by NSF Grants DGE-154362, CNS-1565314, and CNS-1838271 and by the U.S. Department of Energy, Office of Science through the Exascale Computing Project (Contract No. 17-SC-20-SC), the Office of Basic Energy Sciences (Contract No. DE-AC-02-76SF00515), and the Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) Program through the FASTMath Institute under Contract No. DE-AC02-06CH11357.
This work was also supported in part by the U.S. Dept. of Energy, Office of Science Graduate Student Research program (Contract No. DE-SC0014664). The SCGSR program is administered by the Oak Ridge Institute for Science and Education (ORISE), which is managed by ORAU under contract number DE-SC0014664. All opinions in this article are the authors' and do not necessarily reflect the policies and views of the DOE, ORAU, or ORISE.
Authors' addresses: T. H. Chang, Dept. of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061 and Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439; email: tchang@anl.gov; L. T. Watson, Departments of Computer Science, Mathematics, and Aerospace and Ocean Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061; email: ltw@cs.vt.edu; J. Larson, Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439; email: jmlarson@anl.gov; N. Neveu, SLAC National Accelerator Laboratory, Menlo Park, CA 94025; email: nneveu@slac.stanford.edu; W. I. Thacker, Winthrop University, Rock Hill, SC 29733; email: thackerw@winthrop.edu; S. Deshpande, Oracle Labs, Belmont, CA 94002; email: shubhangid@gmail.com; T. C. H. Lux, Dept. of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061; email: tchlux@vt.edu.

0098-3500/2022/09-ART36 $15.00
https://doi.org/10.1145/3529258

Additional Key Words and Phrases: Multiobjective optimization, blackbox optimization, trust region methods, response surface methodology, adaptive weighting schemes

ACM Reference format:
Tyler H. Chang, Layne T. Watson, Jeffrey Larson, Nicole Neveu, William I. Thacker, Shubhangi Deshpande, and Thomas C. H. Lux. 2022. Algorithm 1028: VTMOP: Solver for Blackbox Multiobjective Optimization Problems. *ACM Trans. Math. Softw.* 48, 3, Article 36 (September 2022), 34 pages.
https://doi.org/10.1145/3529258

1 INTRODUCTION

Multiobjective optimization problems (MOPs) arise in many disciplines of science and engineering and are prevalent in the field of multidisciplinary design optimization [60]. In such problems, a multidisciplinary team of engineers must collaborate to design a large complex system, such as an aircraft, balancing design tradeoffs spanning different engineering subfields. In this context, the solution to the MOP is a multidimensional tradeoff surface, called the *Pareto front*, describing the interactions between several conflicting design criteria. Understanding the shape of the Pareto front allows the design engineers to make an informed decision about how to balance design tradeoffs *a posteriori* based on the results of the design optimization. Specific examples of MOPs in engineering are described by [19], who present a ship-hull design problem, and by [67], who describe a chemical engineering problem.

The standard form for a MOP is a real vector-valued minimization problem. The multiobjective cost function F is of the form $F : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X} \subset \mathbb{R}^d$ is the *feasible design space* and $\mathcal{Y} \subset \mathbb{R}^p$ is the *feasible objective space*. For $y, z \in \mathbb{R}^d$, the notation $y < z$ indicates that y is componentwise less than z, and the notation $y \leqq z$ indicates that y is componentwise less than or equal to z, allowing for the possibility that $y = z$. This article considers MOPs where $\mathcal{X}$ is a *simply bounded set*, meaning that $\mathcal{X} = [L, U] = \{x \in \mathbb{R}^d : L \leqq x \leqq U\}$ for some $L, U \in \mathbb{R}^d$ with $L < U$. Then $\mathcal{Y}$ is the image of $[L, U]$ under F. Conceptually, F can be decomposed into p scalar cost functions $f_i : \mathcal{X} \to \mathbb{R}$, $i = 1, \ldots, p$ such that $F(x) = (f_1(x), \ldots, f_p(x))^\top$.

The solution to a MOP is defined by the partial ordering $\leq$ on $\mathcal{Y}$. For $Y, Z \in \mathcal{Y}$, $Y \leq Z$ if Y is componentwise less than or equal to Z, with strict inequality in at least one component. An objective value $F(x^*)$ is said to be *nondominated* if $F(x) \not\leq F(x^*)$ for all $x \in \mathcal{X}$. If $F(x^*)$ is nondominated, then x^* is said to be *efficient*, and the pair $(x^*, F(x^*))$ is *Pareto optimal*. The Pareto front is given by the set of all nondominated objective points and often is accompanied by the set of all efficient designs, called the *efficient set*. The Pareto front has dimension at most $p - 1$ and can be nonsmooth or even discontinuous. Further reading on MOPs can be found in [33]. To discuss the convergence of algorithms, it is convenient to use some additional terminology. In particular, a point $x^\dagger$ is *locally Pareto optimal* if there is no y in a neighborhood of $x^\dagger$ such that $F(y)$ dominates $F(x^\dagger)$. Additionally, when discussing the solutions returned by a multiobjective solver, the term *approximately nondominated* may be used to indicate an objective value $F(\hat{x})$ that is not dominated by any other objective points evaluated by the solver. Then $\hat{x}$ is *approximately efficient*, and $(\hat{x}, F(\hat{x}))$ is *approximately Pareto optimal*.

Remark 1.1. When discussing MOPs, one should distinguish the design and objective spaces. In this article, x, y, and z denote points in the design space, and X, Y, and Z are reserved to denote points in the objective space (or a projection of the objective space, as described in Section 3.1). Each of these entities may be annotated with a superscript to index points or vectors from a set or sequence or with a subscript to denote the components of a point or vector.

This article focuses on *blackbox* MOPs. A process, such as the objective function F, is said to be a blackbox when it can be evaluated at specified inputs to obtain outputs, but the process provides no additional information that is not immediately available in its output. Often, blackbox functions are the result of physical experiments or computationally expensive numerical simulations that require substantial computing resources and time for each evaluation. In some cases, a blackbox multiobjective function F could imply that each component function f_i is an individual blackbox, which must be evaluated separately at a great computational expense. In other cases, all components of F may be obtained through a single integrated simulation.

This article presents the VTMOP software package for solving computationally expensive blackbox MOPs subject to simple bound constraints. Distinctive features of VTMOP are that it (1) handles any $p \geqq 2$, (2) does not require derivative information, and (3) devotes considerable effort to produce well-distributed points over the Pareto front. VTMOP also features a solver subroutine that can be used to solve MOPs when F is available as a Fortran callable function; a "return-to-caller" interface that can be used when F must be decoupled from the optimization algorithm; both parallel and serial execution paradigms; a checkpointing system for storing and recovering function evaluation and iteration data; and support for user-defined local optimization and surrogate modeling procedures.

The article is organized as follows: Section 2 provides some background on **multiobjective optimization algorithms (MOAs)** and widely distributed software packages. Section 3 outlines the VTMOP algorithm and worker subroutines. Section 4 presents two interfaces for solving blackbox MOPs using VTMOP: a return-to-caller interface and a driver subroutine, both of which use the worker subroutines outlined in Section 3. Section 5 is a brief summary of the VTMOP users' manual. Section 6 provides performance results for VTMOP on several analytic test problems. Finally, Section 7 demonstrates VTMOP's performance on a real-world particle accelerator optimization problem.

2 ALGORITHMS AND SOLVERS FOR MULTIOBJECTIVE OPTIMIZATION

This section reviews MOAs and techniques to solve blackbox MOPs of the form

$$\min_{x \in [L,U]} F(x), \tag{1}$$

where the minimization is understood as Pareto optimality. In general, the solution to Equation (1) can be an uncountably infinite set, described by a $(p-1)$-dimensional manifold in the objective space. (When the Pareto front has dimensionality less than $p-1$, this indicates that at least one objective could be eliminated.) The MOAs presented in this section produce a set of approximately nondominated objective points (and corresponding approximately efficient designs), which provide a discrete approximation to the Pareto front.

2.1 Multiobjective Optimization Techniques and Algorithms

Three fundamentally different approaches can be used to solve MOPs. The first approach consists of *a priori* methods, where the decision makers are able to express some preference about the tradeoff **before** viewing the results of the optimization. In these cases, the decision-maker typically supplies a preference function that reduces the MOP to a scalar optimization problem. The second approach consists of *a posteriori* methods, where the decision-maker is not able to express any preference until **after** viewing the results of the optimization procedure. This class is the most general class of methods and the subject of this article. The third class consists of "human-in-the-loop" algorithms, where the decision-maker is able to progressively provide feedback on preference

while the algorithm is running. A summary of all three techniques can be found in [51]; a detailed survey of *a posteriori* techniques for computationally expensive blackbox MOPs is given by [39].

A standard *a posteriori* technique for solving blackbox MOPs is to use multiple *scalarization* functions. Scalarization reduces a MOP to a scalar optimization problem by composing a scalarization function $G : \mathbb{R}^p \to \mathbb{R}$ with F. The resulting function $G \circ F$ can be minimized by using a scalar blackbox optimization solver to find a single (approximately) nondominated point. Optimizing different scalarizations can produce a discrete set of approximately nondominated points, thus approximating the Pareto front. Further reading on general scalarization schemes can be found in Chapter 4 of [33]. One common scalarization scheme is the weighted sum method, which uses a vector of nonnegative weights $w = (w_1, \ldots, w_p)$ to produce a "weighted average" cost function

$$G_w\big(F(x)\big) = \sum_{i=1}^{p} w_i f_i(x). \tag{2}$$

When $w > 0$, any minimizer of Equation (2) is Pareto optimal. However, if any of the component functions f_i are nonconvex, then it is possible that not every point on the Pareto front can be obtained by solving Equation (2) for any vector of weights w. Furthermore, naïvely minimizing Equation (2) using weights that are spaced equally tends to produce clusters of objective values and can leave gaps along the Pareto front. These give the decision-maker a poor understanding of its true shape. Therefore, effective weighted sum scalarization depends on an *adaptive weighting scheme*. Further reading on weighted sum scalarization can be found in Chapter 3 of [33].

One issue with scalarization techniques is that each subproblem must be solved individually as a blackbox scalar optimization problem. Therefore, the cost of solving the MOP is many times that of solving each scalar subproblem. When F is computationally expensive to evaluate, this can be prohibitive. The **response surface methodology (RSM)** can be used to mitigate these expenses [53]. In the multiobjective RSM, a computationally cheap (to evaluate) surrogate function is fit to each objective function by using values from evaluating the objective at experimental design points. Then multiple scalarizations of these surrogates are optimized, each producing a candidate design point. If the design of experiments is thorough and the surrogates are accurate, then evaluating these candidate designs will produce numerous points near the Pareto front. Thus, many approximate Pareto points can be found for little more than the cost of evaluating a single experimental design.

In modern settings, evolutionary MOAs [1] are perhaps the most widely used class of MOAs. These heuristic algorithms extend evolutionary algorithms to the multiobjective case by using a fitness sorting criterion based on the partial ordering $\leq$. An *elitist* evolutionary MOA also maintains previously observed nondominated solutions, using them to ensure that the region of the objective space that is dominated by the kth generation is a superset of the dominated region after the $(k-1)$th generation. Perhaps the most well-known example of an evolutionary MOA is the **nondominated sorting genetic algorithm (NSGA-II)** [29], an elitist evolutionary MOA with low iteration complexity. NSGA-II tends to have poor performance when the number of objectives is large, so it has an extension NSGA-III [16] that takes a more *a priori* approach by attempting to minimize along user-given reference directions. Evolutionary MOAs have an advantage over most scalarization schemes in that they are capable of producing solution points over the Pareto front in each generation. However, the function evaluation budget requirement for evolutionary MOAs tends to be too large for them to be used for computationally expensive problems. For the test problems presented in [30], the recommended budget for an evolutionary algorithm to solve problems with 5–20 design variables and 3 objectives is between 20,000 and 50,000 function evaluations. For a computationally expensive problem where each function evaluation could require minutes, hours, or days and

occupy massive parallel resources, requiring 20,000 function evaluations would be prohibitive. For computationally expensive problems, evolutionary MOAs are often combined with the RSM to reduce computational expense, as described in [34] and [52].

Another class of algorithms is direct search MOAs that extend scalar direct search methods. One example of direct search MOAs consists of multiobjective extensions of the scalar global optimization algorithm **DIviding RECTangles (DIRECT)** [40]. DIRECT is globally convergent for nonconvex functions if the objective function is Lipschitz continuous with a Lipschitz constant that is bounded in the feasible design space $\mathcal{X}$. **Multiobjective DIRECT (MODIR)** [19] extends DIRECT to the multiobjective case by defining and subdividing *potentially Pareto optimal* hyperintervals. Multiobjective DIRECT algorithms of this form offer similar convergence guarantees as Jones' DIRECT when every component function of the objective is Lipschitz continuous [50]. However, in practice, many function evaluations are needed to achieve high-quality solution sets, so [19] recommend combining MODIR with a faster locally convergent MOA, such as the derivative-free multiobjective line search algorithm of [49]. Another example of direct search MOAs are the extensions of **mesh adaptive direct search (MADS)** [7] to MOPs. These techniques combine a global search step with a local polling strategy, which drives convergence to locally Pareto optimal points. In general, these algorithms converge to points that are locally Pareto optimal even for nonsmooth component functions, and they may even converge to a true Pareto point in the limit, depending on the choice of search step. Notable examples of multiobjective MADS algorithms include the biobjective algorithm BiMADS [8] and the algorithm MultiMADS [9], both of which formulate scalarizations of the problem to attempt to produce evenly spaced points on the Pareto front. More recent MADS extensions, such as DMulti-MADS [14] have avoided explicit scalarization by iterating on subsets of the nondominated poll points. Finally, **Direct MultiSearch (DMS)** [28] and MultiGLODS [27] are other notable direct search techniques, which also use polling to drive local convergence, but do not specifically extend the MADS polling strategy.

VTMOP implements the MOA of [32], which combines an adaptive weighting scheme for weighted sum scalarization, RSM, and trust-region methods, the latter of which are widely used in scalar optimization [56]. In the context of the RSM, each **local trust region (LTR)** defines a region of the design space in which the current response surface approximation can be used as a surrogate. In the context of MOPs, these LTRs can be used to control the spacing of points on the Pareto front and force solutions in nonconvex portions of the Pareto front when combined with a weighted sum scalarization approach [59]. [64] show that the technique of finding Pareto minima of surrogates within a sequence of LTRs converges to a locally Pareto optimal point, given that the surrogates' accuracies are improving. [32] rely on a global search via DIRECT followed by RSM inside LTRs to produce subsequences of converging solutions. Therefore, this MOA's convergence is largely driven by the convergence of the global search strategy and the accuracy of the surrogates. As discussed in Section 3, VTMOP is designed to allow for various search strategies, LTR optimizers, and surrogate procedures. However, for the techniques suggested by [32], Lipschitz continuity of all f_i will guarantee convergence.

2.2 Multiobjective Optimization Software

Relatively few of these techniques and algorithms are available in widely distributed software packages for solving computationally expensive MOPs. For such problems, many simulation packages support multiobjective design optimization by solving a single scalarized subproblem, producing a single Pareto optimal design. For example, NASA's FUN3D package performs fluid dynamics-based design simulations and provides support for multiobjective analyses by using the design drivers KSOPT, PORT, or SNOPT to solve a single scalarized subproblem, as described in Section 9.9 of [13]. This approach falls under the *a priori* techniques mentioned in Section 2.1. While this

approach is reasonable for the extremely limited evaluation budget of most computational fluid dynamics-based analyses—[13] recommend a budget of just 20 simulations—it results in no understanding of the complex design tradeoffs and is less general than the *a posteriori* approaches favored here.

With *a posteriori* methods, the following options are available: An implementation of BiMADS is available in the widely used package NOMAD v. 3 [45]. This implementation can be used for constrained, unconstrained, mixed integer, and continuous problems with $p = 2$ objectives, but, since BiMADS only targets biobjective problems, this does not extend to $p > 2$ objectives. For $p > 2$ objectives, MATLAB implementations of DMS and MultiGLODS are available in the BoostDFO MATLAB toolbox [61], including a parallel implementation of both. A Fortran 90 implementation is available for MODIR, which is designed to use a user-specified portion of its function evaluation budget to improve MODIR's solutions via a multiobjective line search. Finally, although there are many widely used implementations of both, the official implementations of NSGA-II and NSGA-III are available within the Python package PyMOO [15].

Several other software packages are worth mentioning, although they are not immediately relevant to this article. The Python package `PyMOSO` [26] solves multiobjective simulation optimization problems on an integer lattice and provides a Python framework for implementing new multiobjective simulation optimization algorithms. Additionally, [3] provide a MATLAB toolbox for applying surrogate modeling to MOPs.

Fitting into the above-mentioned landscape of available multiobjective software, VTMOP provides a robust framework and solver for bound-constrained blackbox optimization problems, with an emphasis on producing evenly distributed Pareto front approximations for $p > 2$ objectives. Some of the techniques used to achieve efficiency and scalability for computationally expensive problems incur a nontrivial iteration cost. Therefore, VTMOP is best suited for MOPs that are computationally expensive to evaluate.

Problems with $p > 2$ objectives (the so-called "many-objective" cases) are generally considered to be significantly more difficult to solve than problems where $p = 2$ because of the relative sparsity of any reasonably sized set of solution points on a high-dimensional Pareto front. In general, other software packages and algorithms for solving these problems either place an additional focus on finding uniformly spaced solution sets or recommend reducing the number of objectives [48]. VTMOP falls into the former of these two categories. NSGA-III [16] is one of the most commonly used examples of the first approach, although it is not a purely *a posteriori* solution, since it depends upon user input of precomputed uniformly spaced reference directions.

Another challenge in many-objective optimization is that of effectively presenting visualizations of the Pareto front approximation to a decision-maker. Addressing this challenge is beyond the scope of this work, but several techniques are described by [65].

3 A FRAMEWORK FOR MULTIOBJECTIVE OPTIMIZATION

The MOA of [32] that is implemented in this article applies the RSM to a sequence of LTRs, each centered at a design point corresponding to an "isolated" objective value from the current set of nondominated points. The algorithm begins from the zeroth iteration, and the kth iteration of the algorithm can be summarized by the following four-step process:

(1) If $k > 0$, then update the current set of nondominated points, and then identify an isolated point $F(\tilde{x}^{(k)})$ in the current set of nondominated points. Update the kth LTR $\Delta^{(k)}$, centering about $\tilde{x}^{(k)}$, and assign the kth set $\mathcal{W}^{(k)}$ of adaptive weights based on objective values near $F(\tilde{x}^{(k)})$. If $k = 0$, then there is no current nondominated set; $\Delta^{(0)} = [L, U]$, and $\mathcal{W}^{(0)}$ is assigned predetermined values.

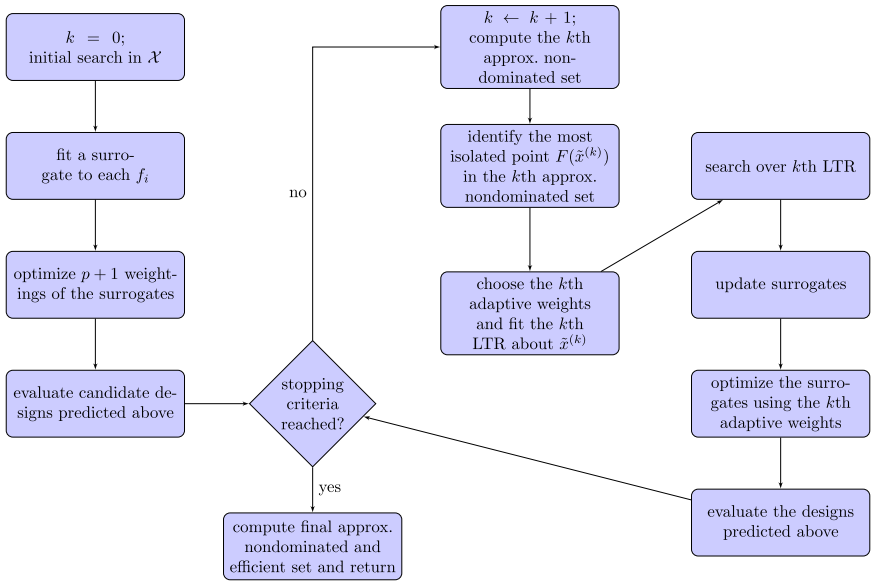

Fig. 1. Algorithm flowchart for VTMOP. For further details on each of these steps, see Sections 3.1, 3.2, 3.3, and 3.4.

(2) Perform the kth exploration by sampling F within $\Delta^{(k)}$; if $k = 0$, then a large exploration budget may be necessary.
(3) Fit p surrogates, $\hat{f}_i^{(k)} \approx f_i$ for $i = 1, \ldots, p$, using the data gathered from Step 2 (plus any data available from previous iterations if $k > 0$). Apply each weight vector in $\mathcal{W}^{(k)}$ to $\hat{f}_1^{(k)}, \ldots, \hat{f}_p^{(k)}$, and minimize the resulting scalarized surrogate problems by using a local optimization strategy. This produces the kth batch $C^{(k)}$ of candidate design points.
(4) Evaluate $F(z)$ for all $z \in C^{(k)}$. Increment k. Update the current database of function values; check for termination conditions; and if no termination conditions have been triggered, proceed to the next iteration.

A flowchart for this process is illustrated in Figure 1. Detailed descriptions of Steps 1–4 are presented in Sections 3.1, 3.2, 3.3, and 3.4, respectively. Note that evaluations of F are required only during Steps 2 and 4.

3.1 Computing the Local Trust Region and Choosing the Adaptive Weights

To construct the kth LTR and select the adaptive weights when $k > 0$, VTMOP must identify an "isolated point" in the current nondominated point set. VTMOP centers the kth LTR at a design point, in the current approximation to the efficient set, whose image is "far away" from any neighbors in the objective space. This enables VTMOP to spend function evaluations where they are needed most and fill in gaps on the current approximation to the Pareto front. When $p = 2$, the Pareto front is a one-dimensional curve, which admits a natural (left-to-right) ordering of points in objective space. [59] identify an isolated point by considering the Euclidean distance from each point in the nondominated set to its left and right neighbors. However, this approach does not generalize to $p > 2$.

[32] generalize this approach to arbitrary dimension using the Delaunay graph. Let $\mathcal{D}^{(k)} = \{x^{(k,1)}, \ldots, x^{(k,n^{(k)})}\}$ denote the set of $n^{(k)}$ design points that have been evaluated at the start of

iteration k, and let $\mathcal{F}^{(k)} = \{F(x^{(k,1)}), \ldots, F(x^{(k,n^{(k)})})\}$. Then the kth approximation to the Pareto front is

$$\mathcal{P}^{(k)} = \left\{ Y \in \mathcal{F}^{(k)} : X \not\leq Y \text{ for all } X \in \mathcal{F}^{(k)} \right\}, \tag{3}$$

and the kth approximation to the efficient set is

$$\begin{aligned} \mathcal{E}^{(k)} = \Big\{ x^{(k,i)} : F\left(x^{(k,i)}\right) \in \mathcal{P}^{(k)} \text{ and } \\ F\left(x^{(k,i)}\right) = F\left(x^{(k,j)}\right) \Rightarrow i \leqq j \text{ for } j = 1, \ldots, n^{(k)} \Big\}. \end{aligned} \tag{4}$$

Remark 3.1. One practicality of many real-world MOPs, especially those with $p > 2$ objectives, is that certain regions of the Pareto front may be fundamentally uninteresting. In particular, decision-makers may be interested in studying tradeoffs only after certain bounds on the objective values have been met. In these situations, the default version of VTMOP may waste precious function evaluations filling in gaps in uninteresting regions of the objective space. VTMOP can account for this by allowing users to provide lower/upper bounds on the range of "interesting" objective values, via optional inputs O_l and O_b. When O_l and O_b are given, Equation (3) is replaced by

$$\mathcal{P}^{(k)} = \left\{ Y \in \mathcal{F}^{(k)} : X \not\leq Y \text{ for all } X \in \mathcal{F}^{(k)} \text{ such that } O_l \leqq X \leqq O_u \right\}.$$

Note that the inclusion of this option does not affect the computation of the final Pareto front approximation, which always includes all observations that are not dominated. Also note that this option must be used with care, since providing overly strict bounds may prevent VTMOP from identifying any points that satisfy them, resulting in early termination. One strategy to circumvent this issue is to run early iterations with relaxed bounds, then tighten the bounds in later iterations after several points have already been found that meet the stricter criteria.

Because the Pareto front is (generically) a $(p-1)$-dimensional manifold, the first step is to project $\mathcal{P}^{(k)}$ into its natural dimension. Let $\mathcal{P}^{(k)} = \{Y^{(k,1)}, \ldots, Y^{(k,m^{(k)})}\}$, where $m^{(k)}$ is the number of unique points in the kth approximately nondominated set; let $Y_1^{(k,i)}, \ldots, Y_p^{(k,i)}$ denote the components of $Y^{(k,i)}$; and let $s^{(k)}$ be a constant such that $s^{(k)} < Y_p^{(k,i)}$ for $i = 1, \ldots, m^{(k)}$. Then the $(p-1)$-dimensional projected set is given by

$$\Pi^{(k)} = \left\{ Z^{(k,1)}, \ldots, Z^{(k,m^{(k)})} \right\}, \; Z^{(k,i)} = \left(\frac{Y_1^{(k,i)}}{Y_p^{(k,i)} - s^{(k)}}, \ldots, \frac{Y_{p-1}^{(k,i)}}{Y_p^{(k,i)} - s^{(k)}} \right). \tag{5}$$

Remark 3.2. To avoid division by zero or a small number when computing Equation (5), the denominators $Y_p^{(k,i)} - s^{(k)}$ must be strictly greater than 0 for all $i = 1, \ldots, m^{(k)}$. In VTMOP, this is done by choosing $s^{(k)} = \min_{Y \in \mathcal{P}^{(k)}} Y_p - 1$.

Remark 3.3. Duplicate values are determined by using distance in the projected space to prevent $\Pi^{(k)}$ from containing duplicate values, which would present problems in the coming computations. Specifically, VTMOP computes Equations (3), (4), and (5) at the same time, and for any $Z^{(k,i)}$ that satisfies $\|Z^{(k,i)} - Z^{(k,j)}\|_2 < \epsilon$, the corresponding values $Y^{(k,i)}$ and $Y^{(k,j)}$ are treated as duplicates. In particular, if $j < i$, then $Z^{(k,i)}$ is omitted from $\Pi^{(k)}$, $Y^{(k,i)}$ is omitted from $\mathcal{P}^{(k)}$, and $F^{-1}(Y^{(k,i)})$ is omitted from $\mathcal{E}^{(k)}$. Here, ϵ is the *objective space tolerance*, a small-scale and machine- and user-preference-dependent constant.

After computing the projected set $\Pi^{(k)}$, its *Delaunay graph* $DG(\Pi^{(k)})$ is used to infer a neighborhood structure. Further details on the Delaunay graph and its computation are provided in

Section 3.1.1. From $DG(\Pi^{(k)})$, an isolation score is computed for each $Y^{(k,i)} \in \mathcal{P}^{(k)}$. Let $\mathcal{N}^{(k,i)} = \{Y^{(k,j)} : Z^{(k,j)}$ be a neighbor of $Z^{(k,i)}$ in $DG(\Pi^{(k)})\}$. Then the isolation score for $Y^{(k,i)}$ is

$$\delta(Y^{(k,i)}) = \sum_{Y \in \mathcal{N}^{(k,i)}} \frac{\|Y^{(k,i)} - Y\|_2}{|\mathcal{N}^{(k,i)}|}. \tag{6}$$

In general, the kth LTR is centered at a design point $\tilde{x}^{(k)}$ that is chosen so $F(\tilde{x}^{(k)}) \in \arg\max_{Y \in \mathcal{P}^{(k)}} \delta(Y)$, whose neighborhood $\tilde{\mathcal{N}}^{(k)}$ is the corresponding $\mathcal{N}^{(k,i)}$. If the Y from the arg max is not unique, then one chooses the $Y^{(k,i)}$ with smallest i.

Remark 3.4. In rare cases, it is possible that $m^{(k)} = 1$. Then $\tilde{x}^{(k)}$ is given by the only element of $\mathcal{E}^{(k)}$, and $\tilde{\mathcal{N}}^{(k)} = \emptyset$.

Let $\tilde{r}^{(k)} \in (0, 1)$ denote the kth LTR radius fraction. Then the kth LTR is given by

$$\Delta^{(k)} = \left[\, \tilde{x}^{(k)} - \tilde{r}^{(k)}(U - L), \tilde{x}^{(k)} + \tilde{r}^{(k)}(U - L) \,\right] \cap \Big[L, U\Big]. \tag{7}$$

Note that $\Delta^{(k)}$ is a simply bounded set.

Remark 3.5. Ideally, performing the kth iteration within $\Delta^{(k)}$ will generate design data in the neighborhood of $\tilde{x}^{(k)}$ and fill in the Pareto front in the neighborhood of $F(\tilde{x}^{(k)})$. When the Pareto front is nonsmooth or discontinuous, however, VTMOP may fail to produce any approximately Pareto optimal points that are near $F(\tilde{x}^{(k)})$ in the objective space. In these cases, $\tilde{x}^{(k)}$ could be a re-occurrence of a previous LTR center. Every time $\|\tilde{x}^{(k)} - \tilde{x}^{(k')}\|_2 < \mu$ for some $k' < k$, the fraction $\tilde{r}^{(k)}$ is decayed using $\tilde{r}^{(k)} = \tau\, \tilde{r}^{(k^*)}$, where $0 < \tau < 1$, and k^* is the largest such k'. Here, μ is the *design space tolerance*, a small-scale and machine-dependent constant. If $\tilde{r}^{(k)}$ drops below some predetermined tolerance, then $\tilde{x}^{(k)}$ is skipped, and the next most isolated point is used instead. For every **new** value of $\tilde{x}^{(k)}$, $\tilde{r}^{(k)}$ is initialized to a default value, regardless of whether the LTR radius has been decayed in previous iterations. The starting trust region radius fraction $\rho^{(0)}$, the decay factor τ, the trust region tolerance $\rho^{(1)}$, and the design space tolerance μ are all optional inputs to VTMOP.

The zeroth iteration is a special case. VTMOP allows users to (optionally) supply a database of previously evaluated designs. However, VTMOP assumes that there is insufficient data available at the start of the zeroth iteration; there can be no $\mathcal{E}^{(0)}$ or $\mathcal{P}^{(0)}$. Instead, $\Delta^{(0)} = [L, U]$, resulting in an exploration of the entire design space. When F is nonconvex, the convergence of VTMOP to the Pareto front is entirely dependent on a thorough design space exploration during the zeroth iteration.

After computing $\Delta^{(k)}$, the kth set of adaptive weights is selected. When $k = 0$ or if $|\tilde{\mathcal{N}}^{(k)}| < 1$, then the adaptive weights are given by $\mathcal{W}^{(k)} = \{e^{(1)}, \ldots, e^{(p)}, \sum_{i=1}^{p} e^{(i)}/p\}$, where $e^{(i)}$ is the ith standard basis vector in $\mathbb{R}^p$. Otherwise, $|\mathcal{W}^{(k)}| = p + |\tilde{\mathcal{N}}^{(k)}|$, and the values of each weight vector are determined as described in Section 3.1.2. The entire process of identifying an isolated point, constructing the kth LTR, and choosing the adaptive weights is summarized in Algorithm 1.

The dominant costs for Algorithm 1 are computing $\mathcal{P}^{(k)}$ using Equation (3) and computing the Delaunay graph $DG(\Pi^{(k)})$ as described in Section 3.1.1. The O time complexity for computing Equation (3) is $O(n^{(k)}m^{(k)})$; and as described in Section 3.1.1, the O time complexity of computing $DG(\Pi^{(k)})$ is a cubic function of $m^{(k)}$ in practice. In practice, the time required by Algorithm 1 is significantly less than the time required to solve the surrogate optimization problems, as described in Section 3.3 and Algorithm 2. A parallel implementation of Algorithm 1 is also briefly described in the beginning of Section 4.3, which further reduces the wallclock time. In the package VTMOP, the subroutine `VTMOP_LTR` implements Algorithm 1.

ALGORITHM 1: Identify an isolated point, construct an LTR, and set adaptive weights.

input
k is the current iteration of VTMOP;
$[L, U]$ is the feasible design space;
$\mathcal{D}^{(k)}$ is the kth design database, containing $n^{(k)}$ design points that were evaluated in previous iterations of VTMOP (if $k = 0$, then $\mathcal{D}^{(0)}$ could contain designs that were evaluated during previous analyses or $\mathcal{D}^{(0)} = \emptyset$);
$\mathcal{F}^{(k)} = \left\{F(x) : x \in \mathcal{D}^{(k)}\right\}$;
$\rho^{(0)}$ is the starting trust region radius (given as a scalar fraction of $U - L$);
$\rho^{(1)}$ is the trust region tolerance (given as a scalar fraction of $U - L$);
τ is the decay factor for the trust region radius;
μ is the design space tolerance;
If $k > 1$, then $\tilde{r}^{(1)}, \ldots, \tilde{r}^{(k-1)}$ are the previous trust region radii fractions;
If $k > 1$, then $\tilde{x}^{(1)}, \ldots, \tilde{x}^{(k-1)}$ are the previous trust region centers;
begin
if $k = 0$ **then**
  $\Delta^{(k)} \leftarrow [L, U]$;
  $\mathcal{W}^{(k)} \leftarrow \left\{e^{(1)}, \ldots, e^{(p)}, \sum_{i=1}^{p} e^{(i)}/p\right\}$;
else
  $\mathcal{P}^{(k)} \leftarrow \left\{Y \in \mathcal{F}^{(k)} : X \not\leq Y \text{ for all } X \in \mathcal{F}^{(k)}\right\} = \left\{Y^{(k,i)} = F\left(y^{(k,i)}\right) \;\middle|\; 1 \leq i \leq m^{(k)}\right\}$ as in (3);
  $\mathcal{E}^{(k)} \leftarrow$ subset of $\mathcal{D}^{(k)}$ as in (4);
  Compute $\Pi^{(k)}$ using (5);
  $m^{(k)} \leftarrow |\Pi^{(k)}|$; **comment** The cardinality $|\mathcal{P}^{(k)}| = |\mathcal{E}^{(k)}| = |\Pi^{(k)}|$ may have changed.
  Compute $DG(\Pi^{(k)})$ (Section 3.1.1);
  for $i = 1, \ldots, m^{(k)}$ **do**
    Compute $\mathcal{N}^{(k,i)}$ based on $DG(\Pi^{(k)})$;
    Compute $\delta(Y^{(k,i)})$ using (6);
  enddo
  initialize $J^{(k)} \leftarrow \left\{1, \ldots, m^{(k)}\right\}$; $\tilde{x}^{(k)} \leftarrow null$;
  while $\tilde{x}^{(k)} = null$ **and** $J^{(k)} \neq \emptyset$ **do**
    $i^* = \min\{\arg\max_{i \in J^{(k)}} \delta\left(Y^{(k,i)}\right)\}$;
    if $\|\tilde{x}^{(k')} - y^{(k,i^*)}\|_2 < \mu$ for any $k' < k$ **then**
      $k^* \leftarrow \max\left\{k' \;\middle|\; k' < k \text{ and } \left\|\tilde{x}^{(k')} - y^{(k,i^*)}\right\|_2 < \mu\right\}$;
      if $\tau\, \tilde{r}^{(k^*)} > \rho^{(1)}$ **then**
        $\tilde{r}^{(k)} \leftarrow \tau\, \tilde{r}^{(k^*)}$;
        $\tilde{x}^{(k)} \leftarrow y^{(k,i^*)}$;
        $\tilde{\mathcal{N}}^{(k)} \leftarrow \mathcal{N}^{(k,i^*)}$;
      else
        $J^{(k)} \leftarrow J^{(k)} \setminus \{i^*\}$;
      endif
    else
      $\tilde{r}^{(k)} \leftarrow \rho^{(0)}$;
      $\tilde{x}^{(k)} \leftarrow y^{(k,i^*)}$;
      $\tilde{\mathcal{N}}^{(k)} \leftarrow \mathcal{N}^{(k,i^*)}$;
    endif
  enddo
  if $\tilde{x}^{(k)} = null$ **then**
    All candidate LTR centers have been maximally refined, which is a termination condition for VTMOP;
    return an appropriate termination message;
  endif
  $\Delta^{(k)} \leftarrow \left[\tilde{x}^{(k)} - \tilde{r}^{(k)}(U - L), \tilde{x}^{(k)} + \tilde{r}^{(k)}(U - L)\right] \cap [L, U]$ as in (7);
  if $|\tilde{\mathcal{N}}^{(k)}| < 1$ **then**
    $\mathcal{W}^{(k)} \leftarrow \left\{e^{(1)}, \ldots, e^{(p)}, \sum_{i=1}^{p} e^{(i)}/p\right\}$;
  else
    Compute $\mathcal{W}^{(k)}$ using (8), as described in Section 3.1.2;
  endif
endif
return $\Delta^{(k)}$, $\mathcal{W}^{(k)}$;

3.1.1 Computing the Delaunay Graph. $DG(\Pi^{(k)})$ is defined in terms of the *Voronoi tessellation* of $\Pi^{(k)}$. The Voronoi tessellation of $\Pi^{(k)}$ is a covering of $\mathbb{R}^{p-1}$ by closed convex polytopes (with disjoint interiors), such that (1) each polytope contains exactly one point in $\Pi^{(k)}$ (this point is called the *Voronoi site* of that polytope) and each point in $\Pi^{(k)}$ is in some polytope interior, and (2) every point in each polytope is as close or closer to its Voronoi site than to any other Voronoi site. The node set for $DG(\Pi^{(k)})$ is given by the set of all Voronoi sites, and the edge list is given by connecting all sites that share a boundary in the Voronoi tessellation.

In the literature, the Delaunay graph is computed as a byproduct of the Delaunay triangulation [18]. However, because of the exponential size of Delaunay triangulations in high-dimensional spaces [41], this approach becomes prohibitively expensive when the number of objectives is of moderate size, for example, when $p > 6$. Another strategy for computing the Delaunay graph is to compute each Voronoi polytope individually [24], allowing for each node's neighborhood to be constructed separately and in parallel. However, this approach ultimately requires more total computations than when constructing the complete Delaunay triangulation, and it is not robust when the input data exhibits geometrical degeneracies.

Instead, VTMOP implements a novel strategy for constructing the connectivity matrix for the Delaunay graph; this strategy is capable of scaling to values of p that are far larger than the number of objectives in any reasonably sized MOP. The proposed Delaunay graph algorithm utilizes the software package DELAUNAYSPARSE [22] to generate a quadratic number of Delaunay simplices, and the expected complexity of this approach is cubic in $m^{(k)}$ and sub-quartic in p [24]. This approach is also highly parallelizable, and across numerous real-world and test problems from the literature, the values of $m^{(k)}$ and p do not grow large enough to constitute a significant expense in VTMOP. Since the details of this Delaunay graph algorithm are novel but not critical to understanding VTMOP, interested readers are directed to the Appendix A.

3.1.2 Choosing the Adaptive Weights. [32] assign a set $\mathcal{W}^{(k)}$ of $p + |\tilde{\mathcal{N}}^{(k)}|$ weight vectors in the kth iteration, given that $k > 0$ and $|\tilde{\mathcal{N}}^{(k)}| > 0$. The first p of these weight vectors are always the standard basis vectors $e^{(1)}, \ldots, e^{(p)}$. By searching for an individual minimum of each component function in every iteration, VTMOP expands the coverage of its approximation to the Pareto front.

The remaining $|\tilde{\mathcal{N}}^{(k)}|$ weight vectors are assigned based on the objective scaling to fill in gaps in the current approximation to the Pareto front. For each $\tilde{Y} \in \tilde{\mathcal{N}}^{(k)}$, the corresponding adaptive weight vector $\tilde{W}$ is given by

$$\tilde{W}_i = \begin{cases} 0, & \text{if} \quad \left| f_i\left(\tilde{x}^{(k)}\right) - \tilde{Y}_i \right| < \epsilon, \\ \dfrac{\tilde{c}}{\left| f_i\left(\tilde{x}^{(k)}\right) - \tilde{Y}_i \right|}, & \text{otherwise}, \end{cases} \tag{8}$$

where $\tilde{c}$ is a normalizing constant such that $\tilde{W}_1 + \cdots + \tilde{W}_p = 1$.

If $k = 0$ or $|\tilde{\mathcal{N}}^{(k)}| = 0$, then there are no "gaps" to fill in. Therefore, VTMOP focuses on expansion, optimizing each individual component function by applying pure weight vectors $e^{(1)}, \ldots, e^{(p)}$. Additionally, an equal weighting of all objectives $\sum_{i=1}^{p} e^{(i)}/p$ is applied.

Remark 3.6. Certain components of $\tilde{W}$ in Equation (8) and the basis vectors $e^{(1)}, \ldots, e^{(p)}$ contain zero values. Recall, however, that minimizing Equation (2) is guaranteed to produce a Pareto optimal point only when $w > 0$. It is impossible to guarantee an exact global solution to Equation (2) in finite time, but it is still desirable that the true solution to each surrogate problem be Pareto optimal. Therefore, after computing the weights in this section, all zero valued components are replaced by a small machine-dependent "fudge factor" and the resulting weights are renormalized. The magnitude of this fudge factor is an optional input to VTMOP.

3.2 Searching the Design Space and Trust Regions

After Algorithm 1, VTMOP performs an exploration of $\Delta^{(k)}$, which is the first step in the multi-objective RSM framework. Depending on whether $k = 0$, $\Delta^{(k)}$ could be either the entire feasible design space $[L, U]$ or the kth LTR, as computed in Equation (7).

Remark 3.7. The number of cost function evaluations used during this exploration should be much larger when $k = 0$ than when $k > 0$. There are two reasons for this. First, the feasible design space could be very large, so a large number of function evaluations may be required for a thorough exploration. Second, when F is nonconvex, the convergence of VTMOP to the Pareto front (under suitable assumptions, such as Lipschitz continuity of each f_i) is entirely dependent on an effective initial search. In future iterations, every function evaluation takes place within an LTR that is centered at a design point whose image is part of the current approximation to the Pareto front.

VTMOP provides two options for the search phase. The first option is an adaptive search technique based on the algorithm DIRECT [40]. This technique is attractive because of the global convergence guarantees of DIRECT, and it often achieves the best performance with a fixed function evaluation budget. The second option is a static search technique that uses a Latin hypercube design to generate a batch of well-distributed design points within the simply bounded search region. This technique does not use function values when generating the design points, thus allowing them to be evaluated concurrently. This can be advantageous in settings where (1) it is convenient to decouple the evaluation of F from VTMOP or (2) enough computing resources are available to support many concurrent evaluations of F [21].

3.2.1 Adaptive Search Using DIRECT. [32] propose using DIRECT to perform an adaptive global search. DIRECT is applied in $\Delta^{(k)}$ at least p times per iteration, once to each component function $f_i(x) = (e^{(i)})^\top F(x)$, with one additional DIRECT search when $k = 0$ using an equal weighting of all the f_is. Specifically,

- if $k = 0$, then use DIRECT to minimize $p + 1$ functions $f_1, \ldots, f_p$ and $\sum_{i=1}^{p} \frac{1}{p} f_i$; and
- if $k > 0$, then use DIRECT to minimize the p component functions $f_1, \ldots, f_p$.

This strategy drives global convergence for Lipschitz continuous f_i (by the original argument of [40]), produces additional data in the neighborhood of the efficient set, and encourages VTMOP to continue expanding its coverage of the Pareto front in iterations with $k > 0$. The implementation of DIRECT in VTDIRECT95 is used for the adaptive search [37].

Remark 3.8. As in Remark 3.6, a fudge factor is applied to each zero valued component of $e^{(1)}$, $\ldots$, $e^{(p)}$, and the weights are renormalized. This approach ensures that each instance of the driver subroutine `VTdirect` from VTDIRECT95 will converge to a Pareto optimal point in the limit.

One important detail of DIRECT is that it samples on an implicit mesh. Since p or $p+1$ instances of the driver `VTdirect` are run in each iteration, this will surely result in redundant design point evaluations. To avoid wasting precious function evaluations, before evaluating F, VTMOP queries its internal database. If the requested design has already been or is currently being evaluated (up to the design tolerance μ), then that design is not re-evaluated, and the corresponding objective value from the database is returned.

Remark 3.9. The number of function evaluations when using DIRECT for a small number of iterations N grows like

$$\Big[(1 - \omega) + \omega(2d + 1)\Big]^{N-1}(2d + 1),$$

where $0 < \omega < 1$ is problem-dependent and typically $\omega \ll 1$. Because DIRECT samples on an implicit mesh, many of these evaluations will be reused between repeated runs in the same search space, so this cost grows sublinearly with the number of objectives.

3.2.2 Static Search Using Latin Hypercube Design. As a static experimental design, the Latin hypercube design is extremely common and is one of the recommendations of [53]. VTMOP uses the subroutine `LATINDESIGN` from the package QNSTOP [4]. Note that, since `LATINDESIGN` is an independent auxiliary subroutine from QNSTOP, this subroutine is extracted from the larger QNSTOP package and included. Latin hypercube designs are stochastic by nature, whereas the search via DIRECT from Section 3.2.1 is deterministic. Redundant design points are an unlikely occurrence when using the Latin hypercube design. To ensure that no function evaluations are wasted, however, VTMOP still checks all design points against its internal database before evaluating.

3.3 Solving the Surrogate Optimization Problem

The kth batch of surrogate optimization problems takes place within $\Delta^{(k)}$. Let $\mathcal{D}^{(k,*)}$ contain all $x \in \mathcal{D}^{(k)}$ plus all the points that were evaluated during the kth search phase from Section 3.2, and let $\mathcal{F}^{(k,*)} = \{F(x) : x \in \mathcal{D}^{(k,*)}\}$. First, p surrogates $\hat{f}_1 \approx f_1, \ldots, \hat{f}_p \approx f_p$ are fit using the data in $\mathcal{D}^{(k,*)}$ and $\mathcal{F}^{(k,*)}$. VTMOP supports the usage of a user-defined surrogate, matching the provided interface. When no user surrogate is supplied, however, VTMOP uses the subroutine `LSHEP` from `SHEPPACK` [63], which implements a linear modified Shepard method. `LSHEP` was chosen as the default surrogate in VTMOP based on `LSHEP`'s demonstrated performance as a surrogate in [31] and the recommendation of [32].

Remark 3.10. A user-defined surrogate is desirable when F is not a true blackbox function and the user is able to exploit domain knowledge to design a better surrogate. For example, if the gradient is available for any f_i, then this information could be used to design a higher-order surrogate. One could integrate such information into the linear modified Shepard approximation by replacing the local linear fit at each data point with the true tangent. For another example, if any component f_i is computationally cheap to evaluate, then surrogate modeling of f_i can be "turned off" by setting $\hat{f}_i = f_i$.

Remark 3.11. It is not uncommon for real-world blackbox MOPs to contain "missing values," often referred to as *hidden constraints* in the literature [46]. These are discrete points in the feasible design space at which F is not defined or produces nonsensical outputs. When such designs are encountered, a missing value is signaled by an `IEEE NAN` value in the output of F. When VTMOP fits the surrogates using the default model `LSHEP`, a wrapper function for the SHEPPACK subroutine is used to place previously evaluated missing values in a separate "`LSHEP` taboo list." Points in the taboo list are not used to fit the surrogates; but if a point in the taboo list is requested while solving the surrogate optimization problem, the output of the surrogates is overwritten by a very large number. This strategy allows VTMOP to converge in the neighborhood of missing values, while explicitly forbidding convergence to a missing value.

After fitting the surrogates, the kth batch of surrogate optimization problems is solved to obtain the kth batch of candidate design points

$$C^{(k)} = \left\{ \operatorname*{arg\,min}_{z \in \Delta^{(k)}} \left[\hat{f}_1(z), \ldots, \hat{f}_p(z)\right]\tilde{W} : \tilde{W} \in \mathcal{W}^{(k)} \right\}. \tag{9}$$

To solve Equation (9), VTMOP allows the usage of a user-defined "local optimization" subroutine. By default, VTMOP uses the polling strategy **Generalized Pattern Search (GPS)** [7]. It should be noted that Equation (9) involves analytic surrogates whose derivative information is available, but

GPS is a polling strategy for blackbox optimization. The rationale for using GPS to solve Equation (9) is as follows:

- GPS polling iterations are locally convergent even for nonsmooth functions, thus improving the robustness of VTMOP.
- When applied to analytic functions, the practical difference between first-order algorithms and blackbox (zeroth order) algorithms is that first-order algorithms typically converge at a faster rate. Solving Equation (9) does not require any evaluations of F, however, so it is an inconsequential expense in the context of computationally expensive blackbox optimization.

The implementation of GPS in VTMOP is designed to be lightweight. In each iteration, GPS polls in every direction along an axis aligned mesh; it steps in the direction of steepest descent, and if no descent direction is found, then it decays the mesh size by a factor of two down to a minimum of μ. The process of fitting the surrogates and solving Equation (9) is summarized in Algorithm 2.

Remark 3.12. VTMOP checks the current database for each new design point before evaluating. In the return-to-caller interface described in Section 4.1, however, the user is required to evaluate $F(z)$ for $z \in \mathcal{C}^{(k)}$. Therefore, VTMOP does not return any redundant or previously evaluated candidate design points (i.e., values in $\mathcal{C}^{(k)}$ that are redundant or equal to values in $\mathcal{D}^{(k,*)}$ up to the design space tolerance μ), which are eliminated from $\mathcal{C}^{(k)}$ in a postprocessing step. In practice, this could mean that $|\mathcal{C}^{(k)}| < |\mathcal{W}^{(k)}|$. This is especially common when F is nonconvex, since different weights in $\mathcal{W}^{(k)}$ could lead to the same solutions in $\mathcal{C}^{(k)}$.

The time complexity of Algorithm 2 is dependent on the cost for fitting the surrogates, the cost for evaluating the surrogates, and the number of surrogate evaluations. By default, VTMOP performs 2,500 iterations of GPS polling (each requiring $2d$ evaluations of the surrogate) per $\tilde{W} \in \mathcal{W}^{(k)}$. In practice, this could require less than a second to several minutes on modern hardware, depending on the size of $\mathcal{W}^{(k)}$, the value of d, and the complexity of the surrogate (`LSHEP`'s evaluation complexity grows sublinearly with $n^{(k)}$). The surrogate problems can also be solved in parallel to reduce the wallclock time of Algorithm 2. In the package VTMOP, the subroutine `VTMOP_OPT` implements Algorithm 2.

3.4 Evaluating Candidate Designs and Iterating

The final step in the kth iteration is to evaluate all $z \in \mathcal{C}^{(k)}$ with F. After all evaluations have finished, the $(k+1)$st databases $\mathcal{D}^{(k+1)}$ and $\mathcal{F}^{(k+1)}$ have been completed. In the next iteration, `VTMOP_LTR` will compute $\mathcal{P}^{(k+1)}$ and $\mathcal{E}^{(k+1)}$, so there is no need to do so in this stage unless a termination condition has been met. The iteration counter k is incremented at this point.

VTMOP has three potential termination conditions. The first condition is that the budget for evaluations of F has been spent. If this limit is reached mid-iteration, then the rest of the iteration aborts. The second condition is that the iteration limit has been exceeded. This condition is checked before the beginning of each iteration. The third condition is that the lowest index point in the inverse image of every point in the current approximation to the nondominated set has been used as the center of a previous LTR, whose trust region radius fraction has been decayed below the tolerance. This condition is checked by `VTMOP_LTR`, as described in Algorithm 1.

When any termination condition occurs at iteration k, the current database of evaluated design points and corresponding function values is used to compute the final nondominated and approximately efficient point sets, which are returned with the termination indicator.

Remark 3.13. Unlike in Section 3.1, upon termination VTMOP returns every observed objective vector that is nondominated and its corresponding efficient design. In particular, these may

ALGORITHM 2: Solve surrogate problems to generate candidate designs.

input
$\mathcal{D}^{(k,*)}$ is a database of evaluated designs, including both $\mathcal{D}^{(k)}$ and additional designs that were evaluated during the kth search (Section 3.2);
$\mathcal{F}^{(k,*)} = \left\{F(x) : x \in \mathcal{D}^{(k,*)}\right\}$;
$\Delta^{(k)}$ is the kth LTR;
$\mathcal{W}^{(k)}$ is the kth set of adaptive weights;
$\hat{f}_1, \ldots, \hat{f}_p$ are user supplied surrogates;
μ is the design space tolerance;
begin
initialize $\mathcal{C}^{(k)} = \emptyset$;
for $i = 1, \ldots, p$ **do**
  Fit $\hat{f}_i \approx f_i$ using $\mathcal{D}^{(k,*)}$ and $\mathcal{F}^{(k,*)}$;
enddo
for $\tilde{W} \in \mathcal{W}^{(k)}$ **do**
  $\tilde{z} \leftarrow \underset{z \in \Delta^{(k)}}{\arg\min} \left[\hat{f}_1(z), \ldots, \hat{f}_p(z)\right]\tilde{W}$;
  if $\|\tilde{z} - x\|_2 > \mu$ for all $x \in \mathcal{D}^{(k,*)}$ **and** $\|\tilde{z} - z\|_2 > \mu$ for all $z \in \mathcal{C}^{(k)}$ **then**
    $\mathcal{C}^{(k)} \leftarrow \mathcal{C}^{(k)} \cup \{\tilde{z}\}$;
  endif
enddo
return $\mathcal{C}^{(k)}$

include duplicate nondominated points. In the literature, these are referred to as the (approximately) weakly nondominated and efficient sets.

4 IMPLEMENTATION AND INTERFACES

VTMOP offers two interfaces for solving blackbox MOPs based on the framework laid out in Section 3. The first is a return-to-caller interface, and the second is the driver subroutine `VTMOP_SOLVE`. In many situations, F can easily be wrapped in a Fortran subroutine, and the driver subroutine is preferred because of its ease of use. However, the return-to-caller interface offers a flexible alternative for situations where the computing environment makes wrapping F in a Fortran subroutine difficult or inefficient. Both interfaces are implemented in ISO Fortran 2008 and support some forms of OpenMP parallelism.

4.1 Return-to-caller Interface

In many real-world blackbox optimization problems, special purpose computing environments [58] and libraries for coordinating the use of parallel resource [21] require the evaluation of F to be decoupled from the optimization algorithm. In these situations, it is convenient to offer a return-to-caller interface, where F is evaluated only from outside of any Fortran subroutine's call stack.

Here, it is necessary to preserve the entire state of the algorithm VTMOP during Steps 2 and 4 (Section 3), where F is evaluated in batches (presumably in parallel). The state is recorded in a Fortran-derived data type `VTMOP_TYPE`, created by the subroutine `VTMOP_INIT`, and stored between invocations of components of `VTMOP`. A user would call `VTMOP_INIT` to create a `VTMOP_TYPE`, then iterate through steps 1–4 from the beginning of Section 3, calling `VTMOP_LTR` for Step 1, `VTMOP_OPT` for Step 3, and performing the evaluations of F in Steps 2 and 4 using their

preferred methods/technologies. Further details on this process are outlined in the USERS document, which is included with the source code. An example implementation of this process is described in [21].

4.2 The Driver Subroutine

The driver subroutine VTMOP_SOLVE implements the same process as in Section 4.1 but without reading and writing the state and using a Fortran implementation of F. The search phase is performed either via DIRECT as in Section 3.2.1 or via Latin hypercube design as in Section 3.2.2, with the choice being specified as an optional input. For a search via DIRECT, the "search budget" inputs are used to specify the iteration limit for VTDIRECT95. For a search via Latin hypercube design, the search budgets are used to specify the number of points in the design.

In addition, VTMOP_SOLVE offers a checkpointing system and a parallel option described in Section 4.3. For further details, see the USERS document, included with the source code.

4.3 Parallel Implementation

Two opportunities for parallelism are offered in VTMOP. First, the iteration tasks in the worker subroutines VTMOP_LTR and VTMOP_OPT can be parallelized. This includes computing $DG(\Pi^{(k)})$ in parallel using the parallel DELAUNAYSPARSEP driver from [22], fitting the p surrogates in parallel, and solving the $|\mathcal{W}^{(k)}|$ surrogate problems (9) in parallel. This iteration level parallelism is provided via OpenMP 4.5 [17] and is available through either the return-to-caller interface or the driver subroutine.

The second source of parallelism is concurrently evaluating F at points requested by VTMOP. In the context of blackbox optimization, this is the more impactful source of parallelism. Since the return-to-caller interface does not evaluate F, this form of parallelism is available only through the driver subroutine (which offers a choice between iteration task parallelism, concurrent function evaluations, or both). Recall that VTMOP requires function evaluations only while searching $\Delta^{(k)}$ as described in Section 3.2 and when evaluating candidate designs from $\mathcal{C}^{(k)}$ as described in Section 3.4. In both cases, concurrent evaluations are achieved by spawning a new OpenMP task for each evaluation of F. Note that because the actual function evaluations are handled by the user, the user is also responsible for capping the number of actual function evaluations to prevent the oversubscription of resources.

Remark 4.1. Within a single evaluation of F, there may be ample opportunities for parallelism, which is problem-dependent and left to the user. In particular, in many real-world applications, each evaluation of F could depend on output from a multinode, distributed simulation. VTMOP does not distribute calls to F, but it does provide the ability for concurrent evaluations of F via OpenMP task-based parallelism. This places the burden of distributing calls to F on the user, but it allows for greater flexibility.

When evaluating $\mathcal{C}^{(k)}$ as in Section 3.4 and when using the Latin hypercube design during the search phase, all candidates/design points can be evaluated in parallel if enough computing resources are available. For a search via DIRECT, the situation is more complicated, since each batch of potentially optimal boxes in a DIRECT iteration can be divided in parallel and each instance of VTdirect can be run asynchronously. VTDIRECT95 offers a parallel driver pVTdirect for distributing the division of boxes, but the distributed memory paradigm is not appropriate for this use case. Instead, a slightly modified implementation of VTdirect, called bVTdirect, is provided, which uses OpenMP tasks to perform concurrent evaluations of box centers. During the kth search, VTMOP makes either $p + 1$ (if $k = 0$) or p (if $k > 0$) asynchronous calls to bVTdirect, resulting in nested parallelism.

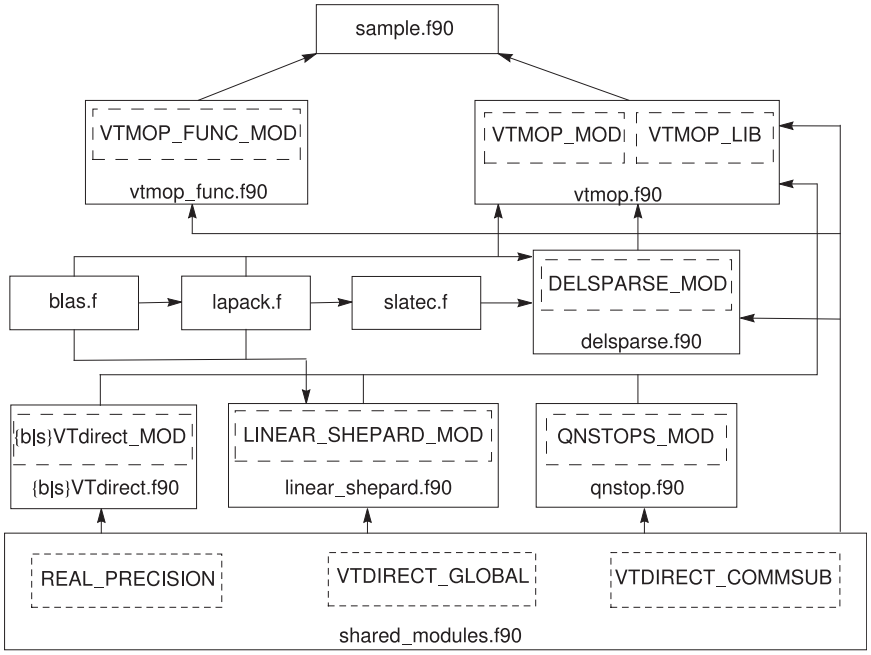

Fig. 2. Dependency graph for VTMOP. Files are outlined in solid lines, and modules are outlined in dashed lines. An arrow from File 1 to File 2 indicates that File 2 has a direct dependency on a subroutine or module from File 1.

Further details on the parallel implementation, such as choosing the number of threads and setting up OpenMP environment variables, are included in the USERS document. A detailed analysis of VTMOP's parallel performance for computationally expensive functions is given in [21].

5 SUMMARY OF USAGE AND PARAMETERS

The package VTMOP is organized as follows: The file `vtmop.f90` contains implementations for the majority of algorithms described in this file, including `VTMOP_LTR` (Algorithm 1), `VTMOP_OPT` (Algorithm 2), `DELAUNAYGRAPH` (Algorithm 3), and `VTMOP_SOLVE` (the driver). Additional files include `vtmop_func.f90` (implementations of test problems from Section 6); `samples.f90` and `samplep.f90` (examples of serial and parallel usage); `sVTdirect.f90` and `bVTdirect.f90` (serial and modified parallel VTDIRECT95 drivers); `shared_modules.f90` (important shared types and modules); `qnstop.f90`, `linear_shepard.f90`, and `delsparse.f90` (minimal copies of dependencies); and `blas.f`, `lapack.f`, and `slatec.f` (minimal copies of common Fortran libraries). The dependency graph for these files is depicted in Figure 2, and additional usage and organization information is provided in the USAGE document.

Remark 5.1. The return-to-caller interface has numerous inputs, outputs, and parameters (many optional) when initializing the data type `VTMOP_TYPE` using `VTMOP_INIT`. Most of these have been discussed in previous sections and remarks, but it is important to understand all of these settings, so they are summarized here. The required inputs are the problem dimensions (d and p) and the bound constraints (L and U). It is impossible to use VTMOP (any interface) without declaring these values. The optional inputs have been carefully assigned default values based on usability and/or trial-and-error experimentation on a wide variety of real-world problems and test problems from the literature. These optional inputs (and their default values) are design space tolerance μ and

objective space tolerance ϵ (by default, the square root of the working precision, defined below); the initial trust region radius $\rho^{(0)}$ as a fraction of $U - L$, the decay rate τ, and the minimum trust region radius $\rho^{(1)}$ as a fraction of $U - L$ (by default, 20% of each dimension of the feasible design space, 0.5, and 10% of the initial trust region radius fraction); the working precision of the machine (by default, the square root of the unit round-off); the fudge factor for zero weights (by default, the fourth root of the unit round-off); the procedures for performing local optimization and the local optimizer iteration budget (by default, `GPS` with a budget of 2,500 iterations); the procedures for fitting and evaluating the surrogates (by default, wrappers for the corresponding linear Shepard's method routines from SHEPPACK); lower and upper bounds on the interesting objective ranges (by default, there are no bounds); a Boolean value specifying whether to perform parallel iteration tasks (by default, false); and the checkpointing mode (i.e., no checkpointing, use checkpointing, or recover from checkpoint, with no checkpointing by default). Every VTMOP subroutine also returns an integer-valued error flag, further detailed in VTMOP's documentation.

Remark 5.2. The driver `VTMOP_SOLVE` accepts some variation of all of the parameters, inputs, and outputs described in Remark 5.1. Additional inputs and outputs include a Fortran implementation of the objective function F, several arrays for returning both the solution sets (approximately Pareto pairs) and full objective function databases, an optional list of precomputed function values, and the budget for the maximum number of blackbox function evaluations. The most notable additional parameter, which will be used in Section 6, is used to specify whether the search via DIRECT should be used (referred to as an adaptive search in the documentation), as opposed to the Latin hypercube search (by default, the search via DIRECT is used). Another optional parameter specifies the search budgets that are used when $k = 0$ and $k > 0$. When using the search via DIRECT, the default values are 10 iterations of DIRECT sampling and 5 iterations of DIRECT sampling, respectively. Otherwise, for a Latin hypercube search, the default values are $16d^2$ and $8d$, respectively.

Remark 5.3. The cost of each search phase can get out of hand if one is not mindful of the interplay between the search budgets and d when using the search via DIRECT. Based on the costs described in Remark 3.9, for large d the Latin hypercube or some incomplete orthogonal design is the better choice.

6 PERFORMANCE ON TEST PROBLEMS

The original algorithm of [32] used a search based on DIRECT. Their approach was previously compared against `BiMADS` from NOMAD, using a budget of only 200 function evaluations [32]. It was also compared against a private implementation of MultiMADS [9] with a budget of 30,000 evaluations of F (as recommended by [30]) for problems with up to six objectives and 36 design variables. In this section, two variations of VTMOP (one using DIRECT for the search and one using a Latin hypercube search) are compared against two widely used open source multiobjective solvers. In particular, the two VTMOP variations are compared against the NSGA-II [29] implementation in PyMOO [15] and an implementation of the direct search technique MultiGLODS [27] from the BoostDFO MATLAB toolbox [61]. Recall that NSGA-II/PyMOO is representative of evolutionary MOA performance, and MultiGLODS is a global multiobjective direct search technique.

Unless otherwise specified, in the remainder of this section, VTMOP is run serially with its default parameter values (specified in Remarks 5.1 and 5.2), using DIRECT (hereafter, referred to as the "DIR variant") and Latin hypercube search (hereafter, referred to as the "LH variant"). NSGA-II is run with PyMOO's default settings: a population size of 100, random sampling for the initial population, a binary tournament selection scheme, simulated binary crossover with probability 0.9, polynomial mutation, and eliminating duplicate values from each population. MultiGLODS is run

with mostly default parameters, most notably, using a global search based on the low-discrepancy Sobol sequence where the number of samples depends on the problem dimension, and evaluating all poll directions in every iteration. One change that was made to the default settings of MultiGLODS was to enable caching of previous function evaluations with a tolerance of 10^{-8}. Note that the DIR variant of VTMOP and the settings for MultiGLODS that use a Sobol sequence search are both deterministic, while the LH variant of VTMOP and NSGA-II are both nondeterministic. Therefore, in all future subsections, the performances of the LH variant and NSGA-II are averaged over five repeated runs.

Evaluation of how well points evaluated by a MOA approximate the true Pareto front is an open problem [6]. In Sections 6.1 and 6.2, three criteria to measure the quality of an approximate Pareto surface are considered:

- the number of nondominated solution points identified,
- the distance between points on the approximate and true Pareto front, and
- the degree to which those points are spread evenly across the entire Pareto front.

The first of these criteria is measured by returning the cardinality of the solution set. The second criterion is measured using the **root mean squared error (RMSE)** between the points in the solution set and their nearest corresponding points on the true Pareto front. This metric can be easily computed only for "nice" test functions, such as those in this section, where the Pareto front can be expressed algebraically. The third criterion is measured using the discrepancy, which is a statistical technique for measuring the uniformity of a sequence of points within some underlying convex space [42, 44]. In this section, the discrepancy of the solution set on the Pareto front is approximated by using the Delaunay triangulation of its projection, as in Equation (5). This measure for spread is fully described in [32] and hereafter referred to as the Delaunay discrepancy. As a discrepancy approximation, the Delaunay discrepancy approaches zero for asymptotically uniformly distributed sequences. Computing the Delaunay discrepancy requires the complete Delaunay triangulation of the projected set, which is obtained using the Quickhull [10] implementation in `SciPy`.

6.1 A Convex Problem

First consider the construction of $F^{(c)}$, a componentwise convex multiobjective function that is meant to capture the performance of MOAs on "nicely behaved" problems. Convex functions are expected to produce easier problems for adaptive weighting schemes, since every point on the Pareto front can be achieved by minimizing a weighted sum of objectives as in Equation (2). Let $d > p$, let $e^{(i)}$ denote the ith standard basis vector in $\mathbb{R}^d$, and let $e = [1, \ldots, 1]^\top$. Then

$$F^{(c)}(x) = \left(\|x - 0.5e^{(1)} - 0.1e\|_2^2, \ldots, \|x - 0.5e^{(p)} - 0.1e\|_2^2 \right), \quad x \in [0,1]^d.$$

The Pareto front for $F^{(c)}$ is a portion of a rotated parabola in $\mathbb{R}^p$.

The two VTMOP variations are compared against NSGA-II and MultiGLODS on $F^{(c)}$ with problem sizes $p = 3, d = 8$ and $p = 4, d = 14$. Performance results for all three metrics are shown in Figures 3 and 4. Note that the number of solution points is shown on a log scale so the results for the smaller budgets can be more easily observed.

As seen in Figures 3 and 4, for the problem $F^{(c)}$, all MOAs considered are able to find tens or even hundreds of approximate solutions with their RMSE less than 0.1 and Delaunay discrepancies of around 0.2 or less within the first 5,000 function evaluations at both problem sizes. Note that it is dangerous to interpret any of the performance measures independently, as none of them improve monotonically. Rather, one might consider whether a large number of solutions with low RMSE and low discrepancy have been observed thus far in each MOA's history, which would indicate that a reasonably dense, well-distributed set of low-error solutions is contained in every subsequent

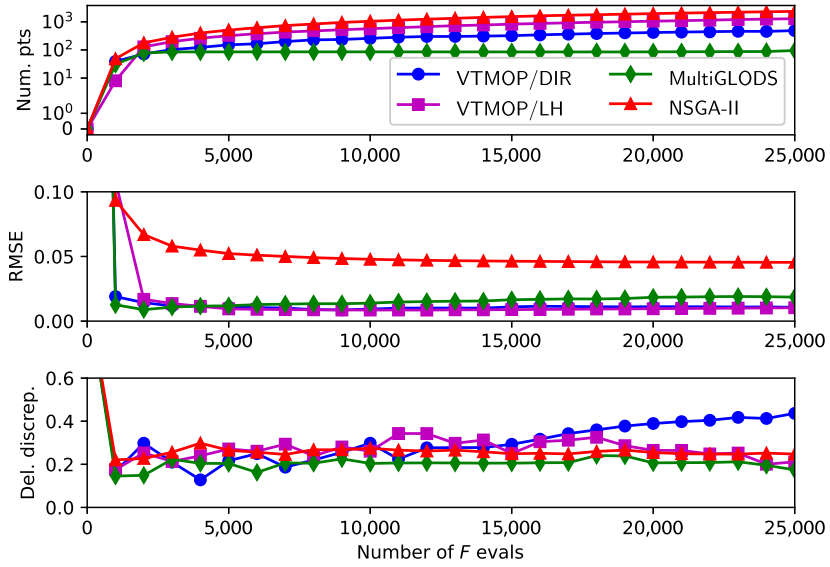

Fig. 3. Performance of MOAs on $F^{(c)}$ for $p = 3$, $d = 8$.

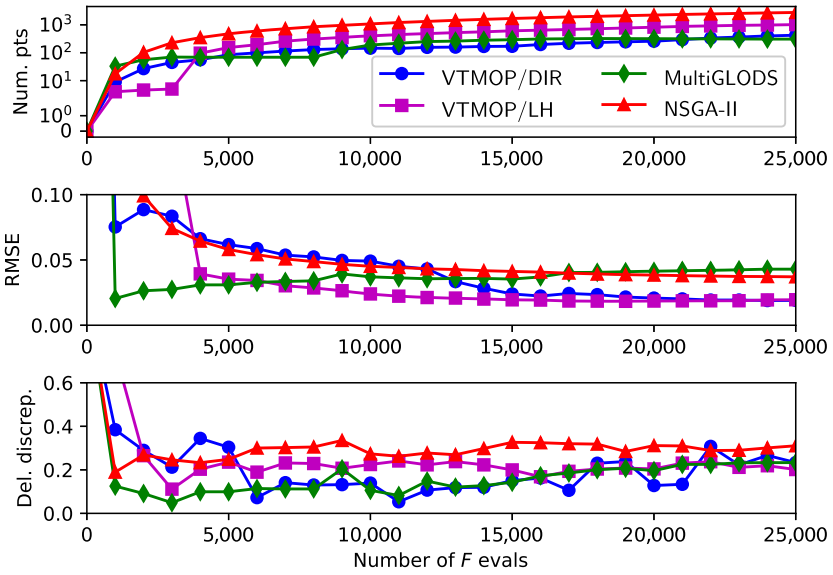

Fig. 4. Performance of MOAs on $F^{(c)}$ for $p = 4$, $d = 14$.

solution set. However, it is interesting to note that both the DIR variant and MultiGLODS tend to produce several low RMSE solutions particularly early on, while NSGA-II and the LH variant tend to produce more solutions in the long term. Also note that the performance statistics are somewhat smoothed for the two stochastic MOAs (LH variant and NSGA-II) due to the averaging effect.

6.2 The DTLZ Problems

The DTLZ test suite [30] is a library of scalable multiobjective test problems that is widely used in the multiobjective optimization literature. Each of the DTLZ problems has a specific property,

which makes it uniquely challenging for MOAs. Additionally, each of the DTLZ problems can be solved with any number of objectives and design variables satisfying $d > p$. In this section, the variations of VTMOP are compared against NSGA-II and MultiGLODS on slight variations of four problems from the DTLZ suite, specifically, DTLZ1, DTLZ2, DTLZ5, and DTLZ7.

One issue with applying the DIR variant directly to the DTLZ problems is that all of the DTLZ problems have their efficient set either located in best- or worst-case locations for DIRECT's deterministic sampling scheme. In particular, for the DTLZ problems, the feasible design space is always the d-dimensional unit hypercube, and for DTLZ1, DTLZ2, and DTLZ5, every efficient x satisfies $x_p = 0.5, \ldots, x_d = 0.5$ (best case for DIRECT), while every efficient x for DTLZ7 satisfies $x_p = 0, \ldots, x_d = 0$ (worst case for DIRECT). Because of the way DIRECT samples, numerous design points x satisfying $x_p = 0.5, \ldots, x_d = 0.5$ will be sampled early, while design points satisfying $x_p = 0, \ldots, x_d = 0$ will be sampled very late. Therefore, to maintain a fair comparison, this section uses variations of the DTLZ problems that have been modified so the efficient set lies in the hyperplane $x_p = 0.6, \ldots, x_d = 0.6$. In most cases, the modification required to achieve this is somewhat obvious. The modified DTLZ problems, along with $F^{(c)}$, are implemented in `vtmop_func.f90`.

The recommended number of design variables for the DTLZ problems range from 5 to 10 more design variables than objectives. The problem sizes of $p = 3, d = 8$ and $p = 4, d = 14$ that were used in Section 6.1 are used again in this section, as they capture the extremes of these recommendations. Since the DTLZ problems are significantly more difficult than $F^{(c)}$, it is not uncommon for MOAs to return solutions that are hugely suboptimal. When the "optimality gap" in a solution point (here quantified by the distance from the solution point to the true Pareto front) is large enough, it can skew performance metrics in a way that is undesirable. Therefore, in this section, only solutions with a distance from the Pareto front of less than 0.1 are considered when computing the performance metrics.

6.2.1 DTLZ1. DTLZ1 has a planar Pareto front, given by all $X \in \mathbb{R}^p$ that satisfy $\sum_{i=1}^{p} X_i = 0.5$. What makes DTLZ1 extremely challenging is that it features $11^{d-p+1} - 1$ "local Pareto fronts," i.e., surfaces of points that are only locally Pareto optimal.

Because of the difficulty of this problem, none of the techniques tested were able to consistently achieve any solutions with a distance from the Pareto front of less than 0.1 for the larger problem size ($p = 4, d = 14$), and only the DIR variant was able to achieve such solutions for the smaller problem size ($p = 3, d = 8$). However, even at the smaller problem size, the DIR variant wasted many iterations due to the fact that Algorithm 1 regularly identifies solutions with an optimality gap greater than 0.1 as the "most isolated point," causing VTMOP to waste many iterations refining in the neighborhood of suboptimal solutions. Therefore, a second run of DIR was performed, specifying "interesting points" (see Remark 3.1) as only those $X \in \mathbb{R}^p$ that satisfy $X \leqq 0.6e$.

Figure 5 shows the results for both runs of the DIR variant on the smaller problem size $p = 3$, $d = 8$ (with and without constraints on the region of interest).

The results in Figure 5 appear inconclusive, since VTMOP achieves a lower RMSE and discrepancy without the objective bounds, but provides more solutions with the objective bounds. Looking at the raw data, it is apparent that VTMOP actually performs significantly better with the objective bounds, however, many of the additional solutions found are clustered and have slightly higher errors, artificially driving up these metrics. One notable observation is that both runs of VTMOP achieved over 100 solutions with RMSE less than 0.07 within the first 2,000 function evaluations.

6.2.2 DTLZ2. DTLZ2 has a spherical Pareto front, given by the portion of the unit sphere that is in the positive orthant. DTLZ2 does not have any points that are locally Pareto optimal that are not globally Pareto optimal; this makes it an easier problem than DTLZ1. However, since the

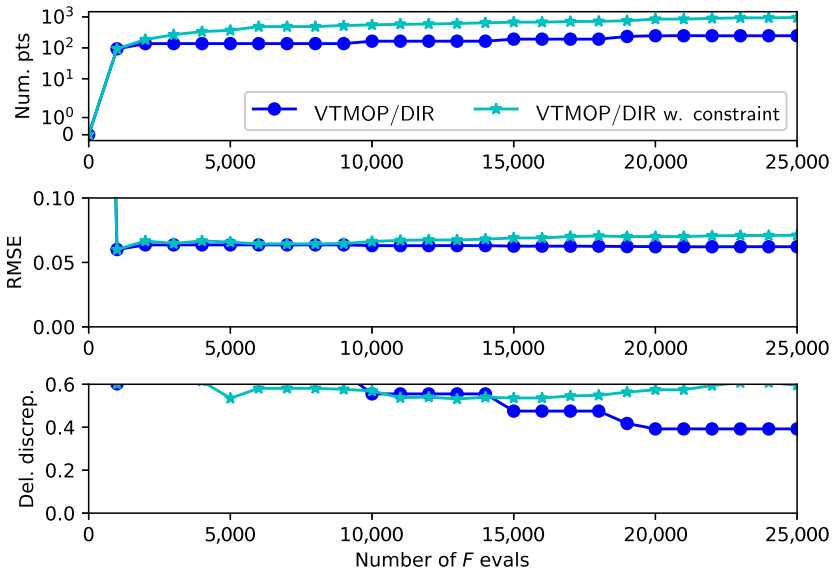

Fig. 5. Performance of VTMOP DIR variant (with and without constraints) on DTLZ1 for $p = 3, d = 8$.

Pareto front for DTLZ2 is entirely concave, it is a particularly challenging problem for adaptive weighting schemes, since only the "pure solutions" (corresponding to basis vectors in $\mathbb{R}^p$) can be obtained as solutions to a weighted sum scalarization, as in Equation (2). The results for solving DTLZ2 with problem sizes $p = 3, d = 8$ and $p = 4, d = 14$ are shown in Figures 6 and 7, respectively.

Similarly as with $F^{(c)}$, all of the MOAs perform well on DTLZ2 at the problem size $p = 3$, $d = 8$. However, in comparison to the results for $F^{(c)}$ in Figure 3, Figure 6 shows fewer solution points found, but those points are generally better distributed on the Pareto front. The fact that fewer solution points are found by VTMOP is somewhat unsurprising, due to the fact that DTLZ2 has a nonconvex Pareto front, which causes different scalarizations to produce the same solution points. However, it may be surprising that VTMOP is not any more affected than the other MOAs, which do not use scalarization schemes. This may be seen as evidence that VTMOP's trust region approach is successfully forcing solutions in these nonconvex regions. The DIR variant again succeeds in producing several low RMSE solutions within the first 5,000 iterations, and in the larger problem, the DIR variant finds several such solutions significantly sooner than all other MOAs except MultiGLODS. However, after quickly discovering these initial solutions, MultiGLODS appears to particularly struggle on the larger problem size ($p = 4, d = 14$) with diversity of solutions, failing to produce just 10 solutions with error less than 0.1 until after 20,000 function evaluations. Due to the lack of repeated trials for MultiGLODS, this could be attributed to a pathological sampling pattern for this problem.

6.2.3 DTLZ5. DTLZ5 has a $(p-2)$-dimensional Pareto front that traces an arc of the Pareto front for DTLZ2. This is considered a degenerate case for MOPs, since many methods expect the Pareto front to be a manifold with dimension $p-1$. For this problem, the Delaunay discrepancy metric cannot be accurately computed, since the Pareto front is not $(p-1)$-dimensional. Therefore, only the number of solutions and RMSE are shown in Figures 8 and 9.

As with DTLZ2, all of the MOAs perform well on the DTLZ5 problems, producing numerous low RMSE solutions with fewer than 5,000 function evaluations. Again, MultiGLODS struggles with the diversity of solutions for the larger problem size ($p = 4, d = 14$), but this may be expected

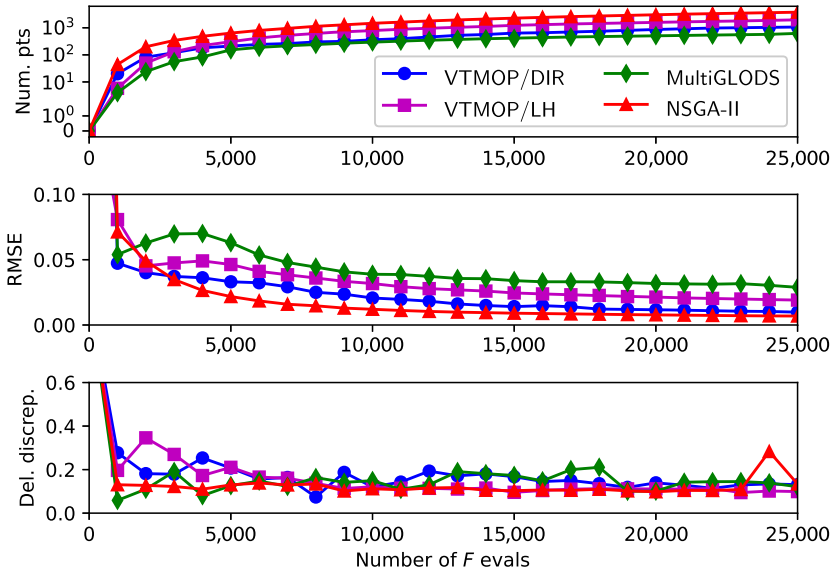

Fig. 6. Performance of MOAs on DTLZ2 for $p = 3$, $d = 8$.

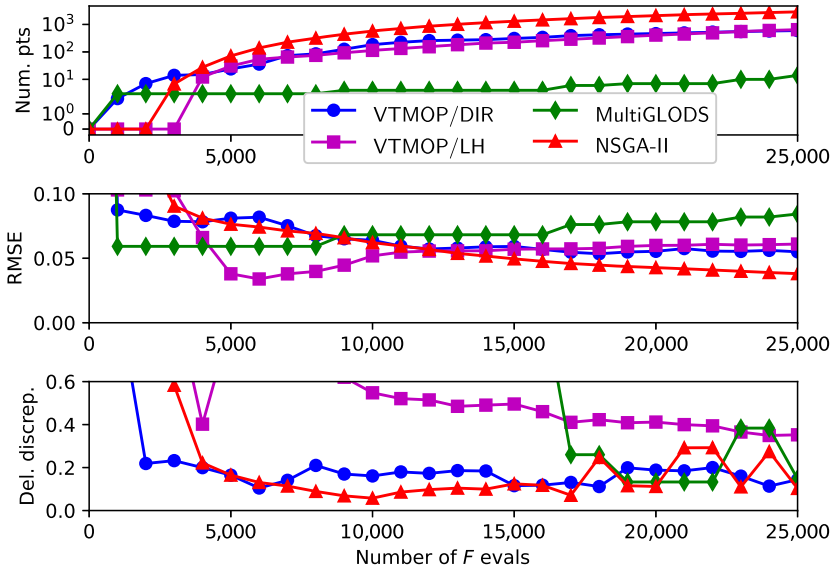

Fig. 7. Performance of MOAs on DTLZ2 for $p = 4$, $d = 14$.

given that DTLZ5 is very similar to DTLZ2 in its nonconvexity. For VTMOP, these problems are a test of robustness, since the $(p-2)$-dimensional Pareto front can cause a degeneracy for DELAUNAYSPARSE when computing the Delaunay graph.

6.2.4 DTLZ7. DTLZ7 has a Pareto front that is discontinuous, consisting of 2^{p-1} disconnected regions. This presents a challenge for techniques such as VTMOP that attempt to fill in gaps in the Pareto front, since it is not possible to achieve a uniform spread of solutions.

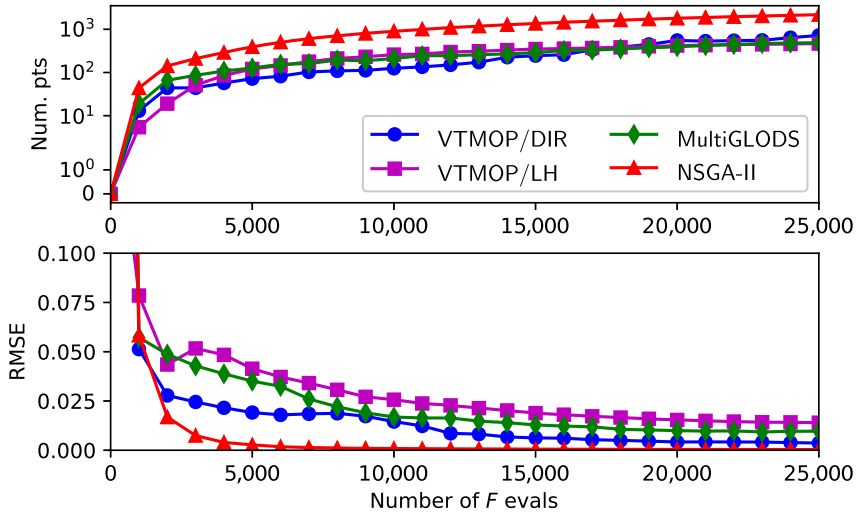

Fig. 8. Performance of MOAs on DTLZ5 for $p = 3, d = 8$.

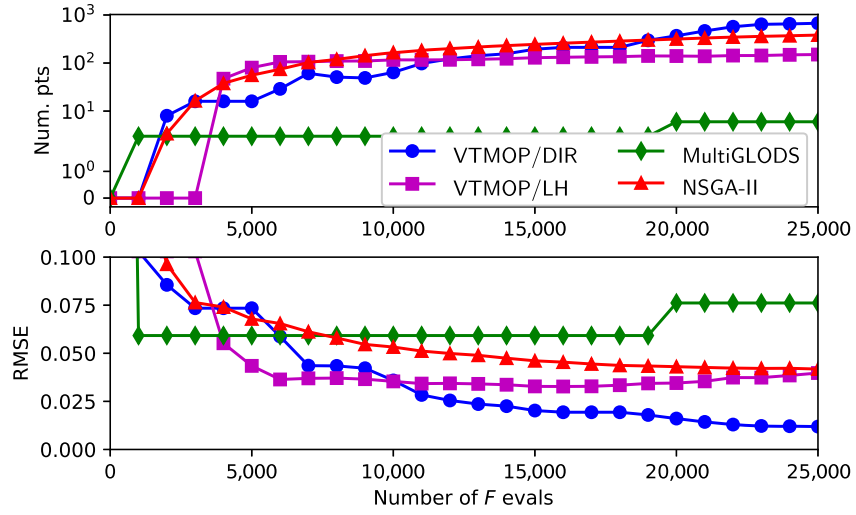

Fig. 9. Performance of MOAs on DTLZ5 for $p = 4, d = 14$.

Similarly to DTLZ1, this problem is particularly challenging for the MOAs considered. None of the algorithms could consistently produce suitable solutions to the larger problem size ($p = 4, d = 14$). However, for the problem size $p = 3, d = 8$, both VTMOP variations and NSGA-II were able to find several solutions. Also, similarly to DTLZ5, the discrepancy metric is not meaningful here, since the Pareto front contains numerous gaps. The results for the smaller problem size are shown in Figure 10.

As seen in Figure 10, NSGA-II appears to perform slightly better than VTMOP on this problem. However, the VTMOP DIR variant is able to discover several low-error solutions within the first 10,000 function evaluations. The VTMOP LH variant is significantly less performant here, requiring many more evaluations to produce just a few solutions. It is worth noting that DTLZ7 is a

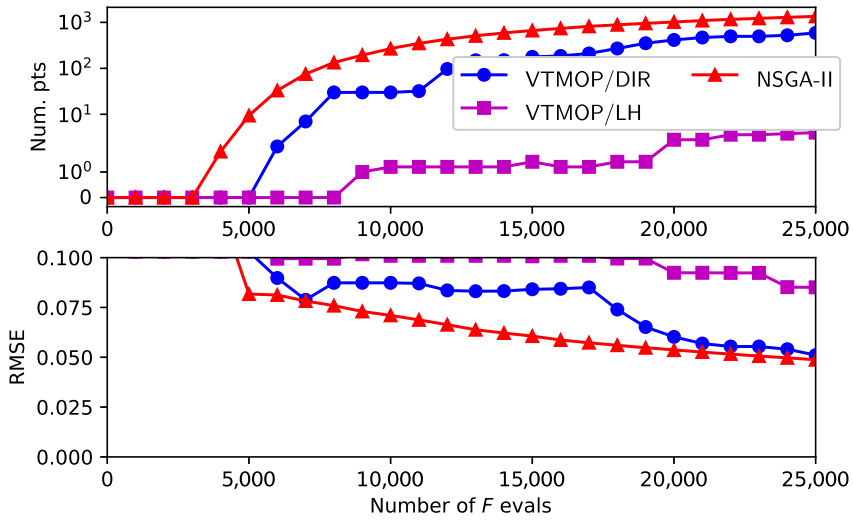

Fig. 10. Performance of MOAs on DTLZ7 for $p = 3$, $d = 8$.

pathological problem for VTMOP, since it is not possible to "fill in" the gaps in DTLZ7's Pareto front, which is the main goal of VTMOP.

6.3 Iteration Time

As noted in Sections 6.1 and 6.2, VTMOP achieves strong performance per number of function evaluations on a variety of test problems. However, this performance comes at an expense. NSGA-II and MultiGLODS spent negligible amounts of time on iteration tasks for all problems, problem sizes, and budgets discussed. However, both variations of VTMOP required significant amounts of CPU time for iteration tasks, especially for larger budgets. This time was largely dominated by the time required to solve the surrogate optimization problems, as described in Section 3.3.

To collect timing results, VTMOP has been run serially on the HPC Bebop at Argonne National Laboratory. For all of the test problems and problem sizes from Sections 6.1 and 6.2, the Linux tool `perf` showed that both the DIR and LH variations spent between 80% and 90% of wallclock time inside the `VTMOP_OPT` subroutine, either fitting the `LSHEP` models or solving the surrogate optimization problems. The majority of remaining time was spent in `VTMOP_LTR` (see Section 3.2), either extracting the Pareto front, calculating the Delaunay graph, or identifying the most isolated point. Note that the majority of these tasks could be parallelized, as described in Section 4.3.

Since the iteration costs are dependent upon the size of VTMOP's internal database, this iteration complexity grows with the total number of function evaluations that have already been performed. In Figure 11, these iteration times are plotted against the total number of function evaluations at the time of the iteration. Note that the y-axis is at a log-scale.

As seen in Figure 11, the iteration times are similar for both and grow very quickly with the size of the database. It appears that the LH variant may require slightly more wallclock time per iteration, but overall, the time requirements are very similar. However, as shown in Figure 12, for the default parameter values (see Section 5), the LH variant will typically complete many more iterations per a fixed function evaluation budget. Therefore, given equal function evaluation budgets, the LH variant will typically require more wallclock time for iterations tasks, overall.

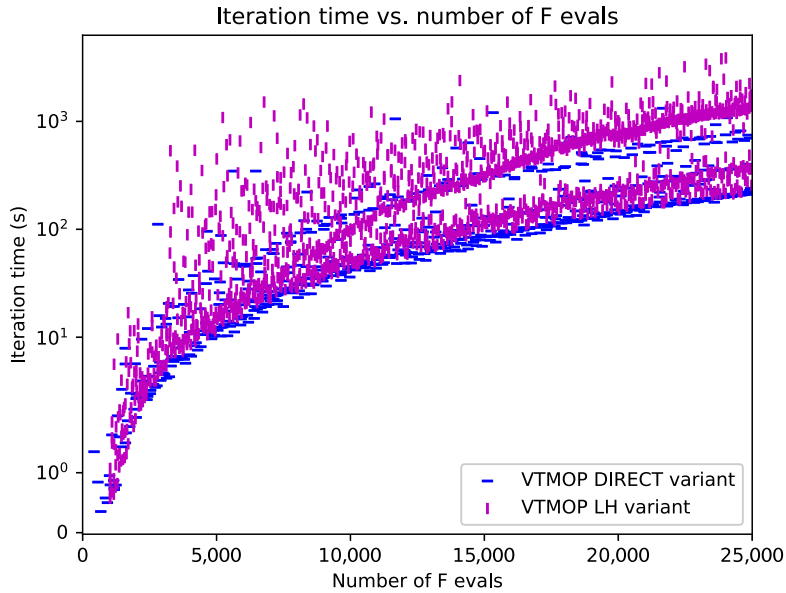

Fig. 11. Iteration time in seconds vs. total number of function evaluations for VTMOP DIR variant and VTMOP LH variant.

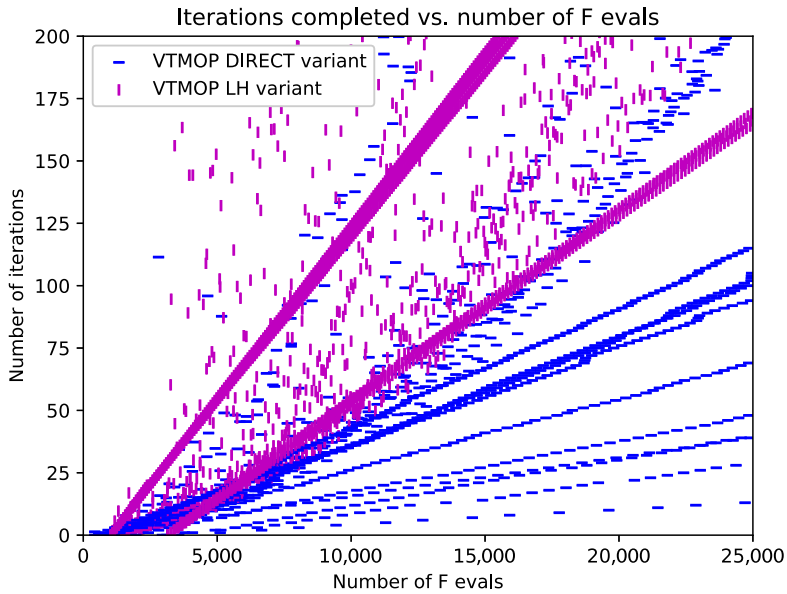

Fig. 12. Number of iterations vs. total number of function evaluations for VTMOP DIR variant and VTMOP LH variant.

6.4 Review of Performance Results

To summarize the performance of VTMOP in Sections 6.1 and 6.2, both variations of VTMOP perform competitively with other state-of-the-art algorithms for large budgets. However, for the larger problem sizes ($p = 4, d = 14$) and harder problems (such as DTLZ1 and DTLZ7), the DIR

variant offers the most consistent results with small function evaluation budgets. In particular, for every problem except DTLZ7 that was solvable by the standards put forth in Section 6.2 (achieving solutions with errors less than 0.1), the DIR variant was able to find numerous solutions with low RMSE within the first 5,000 function evaluations or less. In the case of DTLZ7, the DIR variant was still able to find numerous solutions within the first 10,000 function evaluations.

The consistent performance of the DIR variant with low function evaluation budgets is important, since VTMOP targets computationally expensive problems (such as the problem introduced in Section 7), where larger budgets are infeasible. While the LH variant was not able to match this performance with small budgets, there may be benefit to the LH variant in certain scenarios, since it offers better opportunity for parallelization [21]. However, there is a cost to this improved performance with small budgets. Section 6.3 shows that VTMOP has significant iteration complexity, in some cases requiring many hours of total iteration time to perform 25,000 function evaluations. However, this iteration complexity is backloaded, because the cost of solving each surrogate problem grows with the complexity of the surrogate, which (in the case of LSHEP) depends on the size of VTMOP's database. For budgets smaller than 5,000, VTMOP's total iteration time typically ranges from several seconds up to several minutes. In conclusion, VTMOP is best suited for computationally expensive problems with restrictive budgets, where its increased iteration complexity becomes insignificant, and its ability to consistently produce numerous high-quality approximate solutions with small budgets is extremely valuable.

7 PARTICLE ACCELERATOR DESIGN OPTIMIZATION WITH VTMOP

In this section, VTMOP's performance is demonstrated on a real-world problem in the field of accelerator physics. For another example of VTMOP's performance on a real-world biobjective problem in the area of autotuning LINPACK, see [20].

Given the large scale of particle accelerators, MOA's have gained popularity in optimizing their design and operation due to MOA's ability to handle high-dimensional problems with many objectives ($p > 2$). While scalarization may be possible, it requires *a priori* knowledge of the output space to produce a well-distributed Pareto front that encapsulates all desired information. During design optimization of new accelerators, little may be known of their output space.

This has been the case for a recent project at SLAC National Accelerator Laboratory, where the **Linac Coherent Light Source (LCLS)** is being upgraded to LCLS-II. Design optimizations were performed using an evolutionary MOA [57] and genetic algorithms, especially those of [29]. Genetic algorithms are the most widely methods in the field of particle accelerator design [11, 12, 35, 36, 38, 43, 62, 66] and are commonly used at SLAC. In targeting commissioning goals for LCLS-II, ongoing optimizations have been performed in a piecewise fashion to reduce the number of free parameters. The same approach is taken in this section, and the optimization problem is limited to the first 15 meters of the accelerator. For greater detail on this accelerator problem, see [55], [54], and [47].

For the following results, $d = 7$ and $p = 3$. The upper and lower bounds for the design variables are given in Table 1, alongside the factors by which certain design variables were rescaled to maintain reasonable step sizes during optimization. The objectives of interest are the emittance (ϵ_x), bunch length (σ_z), and energy spread (dE), all of which are to be minimized.

An electron bunch in an accelerator is commonly described with six dimensions: three spatial and three momenta. Emittance is defined as the combined spatial and momentum distribution of the particles in the six-dimensional phase space. Due to symmetries in the problem, the four transverse dimensions are monitored with ϵ_x, the combined spatial and momenta distribution in x. Free electron laser facilities are always in search of improved emittance values, as it directly translates to better X-ray performance downstream. Given this information, VTMOP is directed to

Table 1. Design Variables and Bounds for the LCLS-II Superconducting Injector

Variable	Minimum	Maximum	Unit	Scale Factor
Buncher phase	−100.0	−10.0	Degrees	1
Solenoid strength(s)	0.02	0.07	T/m	1,000
Cavity gradient	0.0	32.0	MV/m	1
Cavity phase	−40.0	40.0	Degrees	1

Two solenoid magnets are included, resulting in a total of seven design dimensions. Note that solenoid strengths were rescaled by the "Scale Factor" before being passed to the optimizer.

focus on the portion of the Pareto front with emittance less than 2 μm (using the optional objective bounds input), as LCLS-II would never be operated with $\epsilon_x > 2\,\mu m$ during normal operations.

The second objective, bunch length, is a metric used to gauge the longitudinal beam size (i.e., the spatial dimension in z). Emittance and bunch length are adversarial objectives that are normally considered when optimizing photoinjectors. The third objective, energy spread, is related to the spread in kinetic energy of the particle distribution. The energy spread is included as a third objective, because previous $p = 2$ optimizations of LCLS-II have been inadequate. While solutions from initial optimization runs produced emittance and bunch length values that met performance metrics, the energy spread was too large and rendered the beam distributions unusable without modification of cavity phases. This is a result of the energy spread's impact on beam behavior in downstream hardware (namely, bunch compressors). This third objective must be accounted for, or the need to manually modify the results after the optimization will remain. In the past, adding a constraint on the energy spread did not provide sufficient improvement in the final optimized beamline, so the choice was made to include it as an additional objective.

It is also necessary to include a penalty term, v, in the objectives to account for infeasible solutions. When a set of parameters results in beam loss, particles are removed from the simulation. If a significant amount of particles are removed, then the emittance, bunch length, and energy spread all can appear artificially small. Given the initial and final number of particles in the simulation (N_0 and N_f), the penalty is calculated as

$$v = \frac{N_0 - N_f}{c},$$

where c is a constant used to scale the penalty to the same order as the scaled objective values. In the case shown in this article, the value of $c = 20.0$ is used. Once the penalty is calculated, it is added to all objective values for that evaluation. The addition of a strictly positive penalty term to all objectives is a natural extension of the augmented Lagrangian approach for nonlinear programming to the multiobjective case [25].

The parallel particle-in-cell code OPAL-T [2] is used to simulate the portion of the LCLS-II accelerator. One OPAL-T simulation of this beamline requires on average 4.3 minutes (with significant variability) on four cores of the HPC Bebop at Argonne National Laboratory. These four-core evaluations are considered low-fidelity simulations. When candidate beamline configurations are identified, higher-fidelity simulations are run to confirm the results and refine working points. Higher-fidelity simulations are too computationally intensive to use within the widespread genetic MOA's that require thousands of function evaluations. Still, even for this lower-fidelity version of the problem, 5,000 evaluations of OPAL-T require well over 1,000 CPU hours on Bebop or over 350 hours of wallclock time when run serially.

VTMOP was applied to this problem with various budgets of OPAL-T evaluations. VTMOP was configured to search using DIRECT and run using its default settings except for the trust region radius fractions, which were reduced to 10% and 2% of the design bounds, respectively, because a

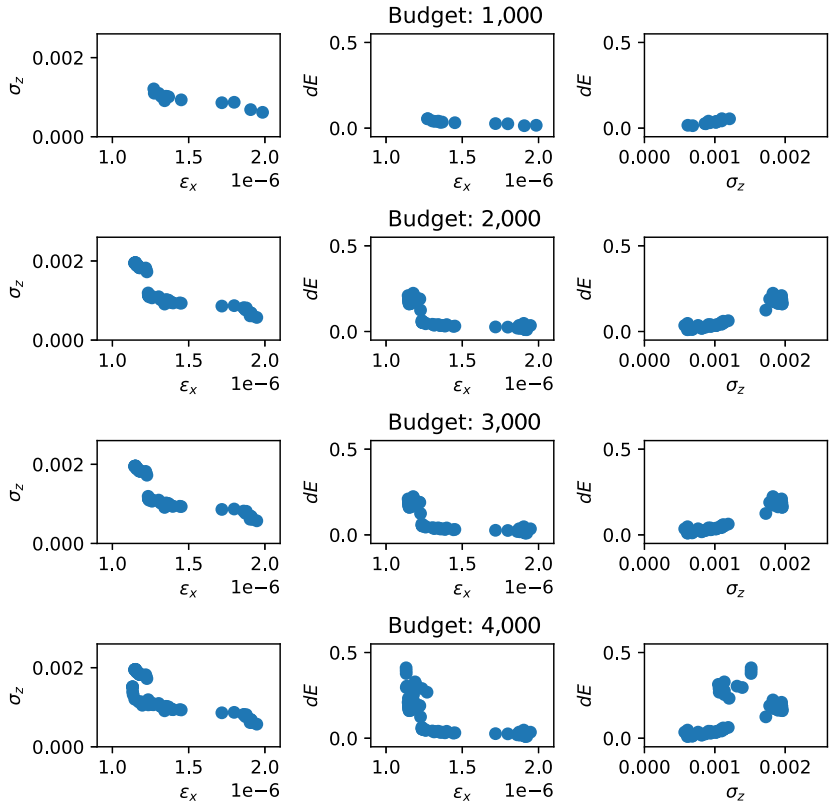

Fig. 13. Biobjective projections of the solutions found by VTMOP for the LCLS-II optimization problem using OPAL-T, after budgets of 1K, 2K, 3K, and 4K function evaluations.

large portion of the bound-constrained design space is actually infeasible for the original problem (resulting in a nonzero penalty v). Additionally, as previously stated, VTMOP was directed to focus on the portion of the Pareto front with emittance less than $2\,\mu m$ using the optional objective bounds input. During initial tests, VTMOP was not able to identify any designs that met this strict objective bound after the zeroth iteration (as discussed in Remark 3.1). Therefore, VTMOP was run for five iterations (1,534 function evaluations) without any bounds on the interesting range of objectives, after which numerous points meeting these bounds were identified. Then the objective bounds were set for $\epsilon_x \leqq 2\,\mu m$ for the remaining 3,466 function evaluations. Finally, to ensure equal treatment of all objectives, in the version of OPAL-T that was run by VTMOP, the raw output values of ϵ_x and σ_z were rescaled so their expected range would match the expected range of dE. Note that this rescaling was only for VTMOP's internal use, and the unscaled values of ϵ_x and σ_z are reported in the following results:

The raw objective values for the solutions found by VTMOP after certain numbers of simulation evaluations are shown in Figure 13. After 4,000 evaluations, no significant further improvements were made. Therefore, to save space, only results up to a budget of 4,000 are shown in Figure 13.

Compared to a typical commissioning optimization of LCLS-II, VTMOP's convergence within 4,000 evaluations is a marked improvement. Previous optimizations of LCLS-II were typically run to 30,000 evaluations with constraints on the objective bounds [54] or 10,000 evaluations with no constraints [47]. In both cases, only the emittance (ϵ_x) and bunch length (σ_z) were considered

($p = 2$), producing solutions with unfit energy spreads (dE). As a result, a significant amount of time was required after the optimization to partially reconfigure the solutions to improve their energy spreads. Note that adding the energy spread as a constraint to NSGA-II has been attempted in the past with poor results. The results shown in Figure 13 indicate reasonable energy spread values for a range of emittance and bunch lengths. This shows an opportunity to simultaneously reduce the number of function evaluations and avoid the need to postprocess solutions after future LCLS-II optimizations, saving a significant amount of time and resources.

APPENDIX

A COMPUTING THE DELAUNAY GRAPH

The pseudocode for the Delaunay graph calculation described in Section 3.1.1 is given in Algorithm 3.

ALGORITHM 3: Compute the Delaunay graph.

input
$\Pi^{(k)} = \left\{Z^{(k,1)}, \ldots, Z^{(k,m^{(k)})}\right\}$ with each $Z^{(k,i)}$ defined as in (5);
begin
$\mathcal{M}^{(k)}$ is a Boolean-valued matrix of dimensions $m^{(k)} \times m^{(k)}$;
initialize

$$\mathcal{M}^{(k)} \leftarrow \begin{bmatrix} FALSE & \ldots & FALSE \\ \vdots & & \vdots \\ FALSE & \ldots & FALSE \end{bmatrix};$$

for $i = 1, \ldots, m^{(k)}$ **do**
  for $j = i + 1, \ldots, m^{(k)}$ **do**
    $\bar{Z}^{(k,i,j)} \leftarrow$ the midpoint between $Z^{(k,i)}$ and $Z^{(k,j)}$;
    $\mathcal{S}^{(k,i,j)} \leftarrow$ the vertex set for a Delaunay simplex containing $\bar{Z}^{(k,i,j)}$;
    if $Z^{(k,i)} \in \mathcal{S}^{(k,i,j)}$ **and** $Z^{(k,j)} \in \mathcal{S}^{(k,i,j)}$ **then**
      $\mathcal{M}^{(k)}_{i,j} \leftarrow TRUE$;
      $\mathcal{M}^{(k)}_{j,i} \leftarrow TRUE$;
    endif
  enddo
enddo
return $\mathcal{M}^{(k)}$

The key cost for Algorithm 3 is computing $\mathcal{S}^{(k,i,j)}$. VTMOP uses the DELAUNAYSPARSE software package [22] to compute each of these simplices. The computational complexity of DELAUNAYSPARSE is $O(m^{(k)}p^2c_f)$ per simplex constructed, where c_f is the number of "flips" required to converge. Empirically, it has been shown that c_f is typically a polynomial function of p and independent of $m^{(k)}$ [23]. Therefore, the execution time for Algorithm 3 is a cubic function of $m^{(k)}$, regardless of the number of objectives p.

Algorithm 3 correctly computes the Delaunay graph when $\Pi^{(k)}$ is in *general position*, meaning that the Delaunay triangulation of $\Pi^{(k)}$ exists and is unique. This is established in Theorem A.1. Existence is guaranteed when $\Pi^{(k)}$ does not lie in a $(p-2)$-dimensional linear manifold, and uniqueness is guaranteed when there is no set of $p + 1$ points in $\Pi^{(k)}$ that lie on the same $(p - 2)$-sphere.

THEOREM A.1. *If $\Pi^{(k)}$ is in general position, then Algorithm 3 computes the connectivity matrix for $DG(\Pi^{(k)})$.*

Proof. Suppose that $\Pi^{(k)}$ is in general position; and let $\mathcal{S}^{(k,i,j)}$, $Z^{(k,i)}$, $Z^{(k,j)}$, and $\bar{Z}^{(k,i,j)}$ be as defined in Algorithm 3. If $\mathcal{S}^{(k,i,j)}$ contains both $Z^{(k,i)}$ and $Z^{(k,j)}$, then clearly $Z^{(k,i)}$ and $Z^{(k,j)}$ are connected in the Delaunay graph. Otherwise, since the simplex whose vertices are $\mathcal{S}^{(k,i,j)}$ contains the midpoint $\bar{Z}^{(k,i,j)}$, at least one facet of this simplex must separate $Z^{(k,i)}$ and $Z^{(k,j)}$. Therefore, $Z^{(k,i)}$ and $Z^{(k,j)}$ cannot be connected in the Delaunay graph. □

Remark A.1. If $DG(\Pi^{(k)})$ is not unique, then Algorithm 3 may fail to produce a connectivity structure that is consistent with any Delaunay triangulation. However, the *Gabriel graph* is a subgraph of every Delaunay triangulation, and therefore the Gabriel graph will always be embedded in the connectivity structure computed by Algorithm 3, regardless of uniqueness.

Remark A.2. If the Delaunay triangulation does not exist, then $\Pi^{(k)}$ is contained in a $(p-2)$-dimensional linear manifold. Typically, this occurs only when $m^{(k)} < p$. Then all points in $\Pi^{(k)}$ are considered neighbors, and the most isolated point corresponds to the projected point that is, on average, farthest away from all other points in $\Pi^{(k)}$. One extremely rare case is that $m^{(k)} \geq p$ but $\Pi^{(k)}$ is contained in a $(p-2)$-dimensional linear manifold due to the Pareto front being a $(p-2)$- or lower-dimensional manifold embedded in the objective space. In this case, a principal component analysis is performed by using the DGESVD subroutine from LAPACK [5] to compute the singular value decomposition of the matrix whose columns are points in $\Pi^{(k)}$ after they have been shifted so their barycenter is the origin. After computing the decomposition, $\Pi^{(k)}$ is projected onto the span of the left singular vectors whose corresponding singular values are greater than the objective space tolerance ϵ (from Remark 3.3). Then $DG(\Pi^{(k)})$ is given by computing the Delaunay graph in this reduced dimensional space.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the computing resources provided on Bebop, a high-performance computing system operated by the Laboratory Computing Resource Center at Argonne National Laboratory. The authors also thank Yuantao Ding at SLAC National Accelerator Laboratory for helpful discussions related to LCLS-II commissioning.

REFERENCES

[1] A. Abraham, L. Jain, and R. Goldberg. 2005. *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications. Advanced Information and Knowledge Processing* series. Springer Verlag, London, UK.

[2] A. Adelmann, P. Calvo, M. Frey, A. Gsell, U. Locans, C. Metzger-Kraus, N. Neveu, C. Rogers, S. Russell, S. Sheehy, J. Snuverink, and D. Winklehner. 2019. OPAL a versatile tool for charged particle accelerator simulations.

[3] A. Al-Dujaili and S. Suresh. 2016. A MATLAB toolbox for surrogate-assisted multi-objective optimization: A preliminary study. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 1209–1216.

[4] B. D. Amos, D. R. Easterling, L. T. Watson, W. I. Thacker, B. S. Castle, and M. W. Trosset. 2020. Algorithm 1007: QNSTOP: Quasi-Newton algorithm for stochatic optimization. *ACM Trans. Math. Softw.* 46, 2 (2020).

[5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. 1999. *LAPACK Users' Guide* (3rd ed.). SIAM, Philadelphia, PA.

[6] C. Audet, J. Bigeon, D. Cartier, and S. Le Digabel. 2021. Performance indicators in multiobjective optimization. *Eur. J. Oper. Res.* 292, 2 (2021), 397–422.

[7] C. Audet and J. E. Dennis. 2006. Mesh adaptive direct search algorithms for constrainted optimization. *SIAM J. Optim.* 17, 1 (2006), 188–217.

[8] C. Audet, G. Savard, and W. Zghal. 2008. Multiobjective optimization through a series of single-objective formulations. *SIAM J. Optim.* 19, 1 (2008), 188–210.

[9] C. Audet, G. Savard, and W. Zghal. 2010. A mesh adaptive direct search algorithm for multiobjective optimization. *Eur. J. Oper. Res.* 204, 3 (2010), 545–556.

[10] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. 1996. The Quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.* 22, 4 (1996), 469–483.

[11] I. V. Bazarov and C. K. Sinclair. 2005. Multivariate optimization of a high brightness DC gun photoinjector. *Phys. Rev. Spec. Topics - Acceler. Beams* 8, 3 (2005).
[12] W. F. Bergan, I. V. Bazarov, C. J. R. Duncan, D. B. Liarte, D. L. Rubin, and J. P. Sethna. 2019. Online storage ring optimization using dimension-reduction and genetic algorithms. *Phys. Rev. Acceler. Beams* 22, 5 (2019).
[13] R. T. Biedron, J. R. Carlson, J. M. Derlaga, P. A. Gnoffo, D. P. Hammond, W. T. Jones, B. Kleb, E. M. Lee-Rausch, E. J. Nielson, M. A. Park, C. L. Rumsey, J. L. Thomas, K. B. Thompson, and W. A. Wood. 2019. *FUN3D Manual: 13.6*. Technical Report Technical Memorandum 2019-220416. NASA, Langley Research Center, Hampton, VA.
[14] J. Bigeon, S. Le Digabel, and L. Salomon. 2021. DMulti-MADS: Mesh adaptive direct multisearch for bound-constrained blackbox multiobjective optimization. *Computat. Optim. Applic.* 79, 2 (2021), 301–338.
[15] J. Blank and K. Deb. 2020. pymoo: Multi-objective optimization in Python. *IEEE Access* 8 (2020), 89497–89509.
[16] J. Blank, K. Deb, and P. C. Roy. 2019. Investigating the normalization procedure of NSGA-III. In *Proceedings of the 10th International Conference on Evolutionary Multi-criterion Optimization*. Springer, 229–240.
[17] OpenMP Architecture Review Board. 2015. *OpenMP Application Programming Interface*. Technical Report version 4.5. OpenMP Architecture Review Board.
[18] J.-D. Boissonnat, O. Devillers, and S. Hornus. 2009. Incremental construction of the Delaunay triangulation and the Delaunay graph in medium dimension. In *Proceedings of the 25th Annual Symposium on Computational Geometry*. ACM, New York, NY, 208–216.
[19] E. F. Campana, M. Diez, G. Liuzzi, S. Lucidi, R. Pellegrini, V. Piccialli, F. Rinaldi, and A. Serani. 2018. A multi-objective DIRECT algorithm for ship hull optimization. *Computat. Optim. Applic.* 71, 1 (2018), 53–72.
[20] T. H. Chang, J. Larson, and L. T. Watson. 2020. Multiobjective optimization of the variability of the high-performance LINPACK solver. In *Proceedings of the Winter Simulation Conference*. IEEE, 3081–3092.
[21] T. H. Chang, J. Larson, L. T. Watson, and T. C. H. Lux. 2020. Managing computationally expensive blackbox multiobjective optimization problems using libEnsemble. In *Proceedings of the 28th High Performance Computing Symposium*. SCS.
[22] T. H. Chang, L. T. Watson, T. C. H. Lux, A. R. Butt, K. W. Cameron, and Y. Hong. 2020. Algorithm 1012: DELAUNAYSPARSE: Interpolation via a sparse subset of the Delaunay triangulation in medium to high dimensions. *ACM Trans. Math. Softw.* 46, 4 (2020).
[23] T. H. Chang, L. T. Watson, T. C. H. Lux, B. Li, L. Xu, A. R. Butt, K. W. Cameron, and Y. Hong. 2018. A polynomial time algorithm for multivariate interpolation in arbitrary dimension via the Delaunay triangulation. In *Proceedings of the ACM Southeast Conference*.
[24] T. H. Chang, L. T. Watson, T. C. H. Lux, S. Raghvendra, B. Li, L. Xu, A. R. Butt, K. W. Cameron, and Y. Hong. 2018. Computing the umbrella neighbourhood of a vertex in the Delaunay triangulation and a single Voronoi cell in arbitrary dimension. In *Proceedings of the IEEE SoutheastCon*.
[25] G. Cocchi and M. Lapucci. 2020. An augmented Lagrangian algorithm for multi-objective optimization. *Computat. Optim. Applic.* 77, 1 (2020), 29–56.
[26] K. Cooper and S. R. Hunter. 2020. PyMOSO: Software for multi-objective simulation optimization with R-PERLE and R-MinRLE. *INFORMS J. Comput.* 32, 4 (2020), 1101–1108.
[27] A. L. Custódio and J. F. A. Madeira. 2018. MultiGLODS: Global and local multiobjective optimization using direct search. *J. Glob. Optim.* 72, 2 (2018), 323–345.
[28] A. L. Custódio, J. F. A. Madeira, A. I. F. Vaz, and L. N. Vicente. 2011. Direct multisearch for multiobjective optimization. *SIAM J. Optim.* 21, 3 (2011), 1109–1140.
[29] K. Deb, A. Pratap, S. Agarwel, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolut. Computat.* 6, 2 (2002), 182–197.
[30] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. 2002. Scalable multi-objective optimization test problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE, 825–830.
[31] L. T. Deshpande, S. Watson, J. Shu, and N. Ramakrishnan. 2011. Data driven surrogate-based optimization in the problem solving environment WBCSim. *Eng. Comput.* 27, 3 (2011), 211–223.
[32] S. Deshpande, L. T. Watson, and R. A. Canfield. 2016. Multiobjective optimization using an adaptive weighting scheme. *Optim. Meth. Softw.* 31, 1 (2016), 110–133.
[33] M. Ehrgott. 2005. *Multicriteria Optimization* (2nd ed.). Springer Science & Business Media, Heidelberg, Germany.
[34] Y. Feng, Z. Chen, Y. Dai, F. Wang, J. Cai, and Z. Shen. 2018. Multidisciplinary optimization of an offshore aquaculture vessel hull form based on the support vector regression surrogate model. *Ocean Eng.* 166 (2018), 145–158.
[35] C. Gulliford, A. Bartnik, and I. Bazarov. 2016. Multiobjective optimizations of a novel cryocooled DC gun based ultrafast electron diffraction beam line. *Phys. Rev. Acceler. Beams* 19, 9 (2016).
[36] C. Gulliford, A. Bartnik, I. Bazarov, and J. Maxson. 2017. Multiobjective optimization design of an rf gun based electron diffraction beam line. *Phys. Rev. Acceler. Beams* 20, 3 (2017).

[37] J. He, L. T. Watson, and M. Sosonkina. 2009. Algorithm 897: VTDIRECT95: Serial and parallel codes for the global optimization algorithm DIRECT. *ACM Trans. Math. Softw.* 36, 3 (2009).
[38] A. Hofler, B. Terzic, M. Kramer, A. Zvezdin, V. Morozov, Y. Roblin, F. Lin, and C. Jarvis. 2013. Innovative applications of genetic algorithms to problems in accelerator physics. *Phys. Rev. Spec. Topics - Acceler. Beams* 16, 1 (2013).
[39] S. R. Hunter, E. A. Applegate, V. Arora, and B. Chong. 2019. An introduction to multiobjective simulation optimization. *ACM Trans. Model. Comput. Simul.* 29, 1 (2019), 1–36.
[40] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. 1993. Lipschitzian optimization without the Lipschitz constant. *J. Optim. Theor. Applic.* 79, 1 (1993), 157–181.
[41] V. Klee. 1980. On the complexity of d-dimensional Voronoi diagrams. *Archiv der Mathematik* 34, 1 (1980), 75–80.
[42] L. Kocis and W. J. Whiten. 1997. Computational investigations of low-discrepancy sequences. *ACM Trans. Math. Softw.* 23, 2 (1997), 266–294.
[43] M. Kranjčević, A. Adelmann, P. Arbenz, A. Citterio, and L. Stingelin. 2019. Multi-objective shape optimization of radio frequency cavities using an evolutionary algorithm. *Nuclear Instrum. Meth. Phys. Res. Sect. A: Acceler., Spectrom., Detect. Assoc. Equip.* 920 (2019), 106–114.
[44] S. C. Kugele, M. W. Trosset, and L. T. Watson. 2008. Numerical integration in statistical decision-theoretic methods for robust design optimization. *Struct. Multidisc. Optim.* 36 (2008), 457–475.
[45] S. Le Digabel. 2011. Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm. *ACM Trans. Math. Softw.* 37, 4 (2011).
[46] S. Le Digabel and S. M. Wild. 2015. *A Taxonomy of Constraints in Simulation-based Optimization.* Technical Report G-2015-57. Les cashiers du GERAD, Montréal, Canada.
[47] R. Lemons, N. Neveu, J. Duris, A. Marinelli, C. Durfee, and S. Carbajo. 2022. Temporal shaping of narrow-band picosecond pulses via non-colinear sum-frequency mixing of dispersion-controlled pulses. *Phys. Rev. Spec. Topics - Acceler. Beams* 25, 1 (2022).
[48] K. Li, R. Wang, T. Zhang, and H. Ishibuchi. 2018. Evolutionary many-objective optimization: A comparative study of the state-of-the-art. *IEEE Access* 6 (2018), 194–214.
[49] G. Liuzzi, S. Lucidi, and F. Rinaldi. 2016. A derivative-free approach to constrained multiobjective nonsmooth optimization. *SIAM J. Optim.* 26, 4 (2016), 2744–2774.
[50] A. Lovison and K. Miettinen. 2021. On the extension of the DIRECT algorithm to multiple objectives. *J. Glob. Optim.* 79, 2 (2021), 387–412.
[51] R. T. Marler and J. S. Arora. 2004. Survey of multi-objective optimization methods for engineering. *Struct. Multidisc. Optim.* 26, 6 (2004), 369–395.
[52] J. Müller. 2017. SOCEMO: Surrogate optimization of computationally expensive multiobjective problems. *INFORMS J. Comput.* 29, 4 (2017), 581–596.
[53] R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook. 2016. *Response Surface Methodology: Process and Design Optimization Using Designed Experiments* (4th ed.). John Wiley & Sons, Inc., Hoboken, NJ.
[54] N. Neveu and Y. Ding. 2021. *LCLS-II Standard Configuration with a Gaussian-profile Injector Laser.* Technical Report LCLS-II-TN-21-01. SLAC National Accelerator Laboratory, Menlo Park, CA.
[55] N. Neveu, N. Sudar, Y. Ding, G. Marcus, A. Marinelli, and C. Mayes. 2020. *LCLS-II Standard Configuration with a Gaussian-profile Injector Laser.* Technical Report LCLS-II-TN-20-03. SLAC National Accelerator Laboratory, Menlo Park, CA.
[56] J. Nocedal and S. J. Wright. 2006. *Numerical Optimization* (2nd ed.). Springer Science & Business Media, Heidelberg, Germany.
[57] J. Qiang, C. Mitchell, and A. Qiang. 2016. Tuning of an adaptive unified differential evolution algorithm for global optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation.* IEEE, Vancouver, BC, Canada, 4061–4068.
[58] C. Raghunath, T. H. Chang, L. T. Watson, M. Jrad, R. K. Kapania, and R. M. Kolonay. 2017. Global deterministic and stochastic optimization in a service oriented architecture. In *Proceedings of the 25th High Performance Computing Symposium.* SCS.
[59] J. Ryu, S. Kim, and H. Wan. 2009. Pareto front approximation with adaptive weighted sum method in multiobjective simulation optimization. In *Proceedings of the Winter Simulation Conference.* IEEE, 623–633.
[60] J. Sobieszczanski-Sobieski, A. Morris, and M. Van Tooren. 2015. *Multidisciplinary Design Optimization Supported by Knowledge Based Engineering.* John Wiley & Sons, Ltd., Chichester, UK.
[61] S. Tavares, C. P. Brás, A. L. Custódio, V. Duarte, and P. Medeiros. 2022. Parallel strategies for Direct Multisearch. Numerical Algorithms (2022). To appear.
[62] B. Terzic, A. S. Hofler, C. J. Reeves, S. A. Khan, G. A. Krafft, J. Benesch, A. Freyberger, and D. Ranjan. 2014. Simultaneous optimization of the cavity heat load and trip rates in linacs using a genetic algorithm. *Phys. Rev. Spec. Topics - Acceler. Beams* 17, 10 (2014).

[63] W. I. Thacker, J. Zhang, L. T. Watson, J. B. Birch, M. A. Iyer, and M. W. Berry. 2010. Algorithm 905: SHEPPACK: Modified Shepard algorithm for interpolation of scattered multivariate data. *ACM Trans. Math. Softw.* 37 (2010).
[64] J. Thomann and G. Eichfelder. 2019. A trust-region algorithm for heterogeneous multiobjective optimization. *SIAM J. Optim.* 29, 2 (2019), 1017–1047.
[65] T. Tušar and B. Filipič. 2014. Visualization of Pareto front approximations in evolutionary multiobjective optimization: A critical review and the prosection method. *IEEE Trans. Evolut. Computat.* 19, 2 (2014), 225–245.
[66] L. Yang, D. Robin, F. Sannibale, C. Steier, and W. Wan. 2009. Global optimization of an accelerator lattice using multiobjective genetic algorithms. *Nuclear Instrum. Meth. Phys. Res. Sect. A: Acceler., Spectrom., Detect. Assoc. Equip.* 609, 1 (2009), 50–57.
[67] A. K. Y. Yee, A. K. Ray, and G. P. Rangaiah. 2003. Multiobjective optimization of an industrial styrene reactor. *Comput. Chem. Eng.* 27, 1 (2003), 111–130.

Received June 2020; revised March 2022; accepted March 2022