ETH Zürich

Master Thesis

A Homotopy Method for Large-Scale Multi-Objective Optimization

Author:
Andrew Foster

Supervisor:
Dr. Andreas Adelmann
Prof. Dr. Peter Arbenz
Yves Ineichen

January 10, 2013

Abstract

A method for multi-objective optimization that produces uniformly sampled Pareto fronts by construction is presented. While the algorithm is general, of particular interest is application to simulation-based engineering optimization problems where economy of function evaluations, smoothness of result, and time-to-solution are critical. The presented algorithm achieves an order of magnitude improvement over other geometrically motivated methods, like Normal Boundary Intersection and Normal Constraint, with respect to solution evenness for similar computational expense. Furthermore, the resulting uniformity of solutions extends even to more difficult problems, such as those appearing in common Evolutionary Algorithm test cases. Finally, we apply the algorithm to a problem in the field of particle accelerator design and present promising strong scalability benchmarks.

Contents

Chapter 1

Background & Theory

The problem of scalar function minimization is almost ubiquitous in modern engineering sciences and a number of algorithms exist that exploit various functional characteristics to reach timely solutions. Far more difficult is the problem of vector function minimization, where a number of different, often conflicting, objectives are optimized in such a way as to strike a balance that pleases the end-user. The presence of multiple output dimensions requires a generalization of our concept of optimality, demands significantly more computational effort, as well as compounds the inherent difficulty of the problem.

1.1 Multi-Objective Optimization

Precisely stated, we seek to address the problem

$$\begin{array}{c} \min_{\mathbf{x}\in D} \mathbf{F}(\mathbf{x}) = [F_1(\mathbf{x}), F_2(\mathbf{x}), ..., F_k(\mathbf{x})]^T \\ \text{subject to} \\ g_j(\mathbf{x}) \leq 0 \quad \forall j \in [1, m] \\ h_l(\mathbf{x}) = 0 \quad \forall l \in [1, e] \end{array} \tag{1.1}$$

where the objective functions, F_i, are bounded and defined over the set $D \in \mathbb{R}^d$. Here, $\mathbf{x}$ is a vector of independent design variables of length d and contained in D. In this formulation, the set of objective functions, $\{F_i(\mathbf{x})\colon \forall i \in [1, k]\}$, form a map

$$\mathbf{F}\colon \mathbb{R}^d \to \mathbb{R}^k \tag{1.2}$$

from design space to objective space. The set of all $\mathbf{x} \in D$ satisfying the inequality constraints, $g_j(\mathbf{x}) \leq 0$, and the equality constraints, $h_l(\mathbf{x}) = 0$,

$$X\colon = \left\{ \mathbf{x} \in D\colon \begin{array}{ll} g_j(\mathbf{x}) \leq 0 & \forall j \in [1, m] \\ h_l(\mathbf{x}) = 0 & \forall l \in [1, e] \end{array} \right\} \tag{1.3}$$

is called the feasible set. It's image in objective space

$$Z\colon = \left\{ \mathbf{F}(\mathbf{x})\colon \mathbf{x} \in D \text{ and } \begin{array}{ll} g_j(\mathbf{x}) \leq 0 & \forall j \in [1, m] \\ h_l(\mathbf{x}) = 0 & \forall l \in [1, e] \end{array} \right\} \tag{1.4}$$

is denoted the attainable set.

The objectives, being bounded, scalar-valued functions over a closed set, each attain a minimum value, F_i^*, for some $\mathbf{x}_i^*$ in the feasible set, called an individual minimizer of F_i. The problem is well-posed in the event that the sets of individual minimizers are mutually non-disjoint. In this case, the solution is simply the intersection of these inidividual minimizer sets, as they permit simultaneous minimization of all objective functions.

The more interesting case, involving at least one pair of mutually disjoint individual minimizer sets, implies that not all objectives can be optimized simultaneously. Therefore, since the space of objective vectors does not inherently specify an order, we can not define a solution without additional conceptual scaffolding.

One simple method of making this problem more tractable is to explicitly impose an order on the objective space. This can be done, for example, by introducing a suitable scalarization function

$$s(\mathbf{F})\colon \mathbb{R}^k \to \mathbb{R} \tag{1.5}$$

projecting the objective space onto, and using the well-ordered property of, the real numbers to define a solution. This effectively reduces the task of (1.1) to a scalar minimization problem.

$$\min_{\mathbf{x}\in X} s(\mathbf{F}(\mathbf{x})) \tag{1.6}$$

This approach, however, requires a specification of relative importance among the objectives, which is implied by the scalarization function. For instance, given two objectives, the scalarization

$$s(\mathbf{F}) = 2F_1 + F_2 \tag{1.7}$$

would lead the minimization procedure to a prefer a unit reduction of F_1 over an equal reduction in F_2. Therefore, parameterizing the scalarization function amounts to an a priori articulation of preference of solution features, restricting our ability to explore the objective space.

1.2 A Posteriori Solutions

A far more flexible approach to obtaining solutions of (1.1) involves fully mapping the transitions between individual minimizers. With this information, we are able to select from a range of potential solutions, thereby eliminating the need for an explicit, user-supplied ranking of objective importance in the minimization procedure. This enables an implicit, a posteriori articulation of preference amongst the objectives and permits a complete probing of the solution space.

We require, however, that the members of this solution set from which we choose satisfy certain criteria, namely, that they, in some sense, minimize $\mathbf{F(x)}$ given ordering criteria imposed on the objective space. The notion of Pareto optimality captures this concept precisely.

Definition 1. $\boldsymbol{x}^* \in X$ *is Pareto optimal* $\iff \nexists \boldsymbol{x} \in X \colon F_i(\boldsymbol{x}) \leq F_i(\boldsymbol{x}^*)$ $\forall i \in [1,k]$ *and* $F_i(\boldsymbol{x}) < F_i(\boldsymbol{x}^*)$ *for some* $i \in [1,k]$.

Loosely speaking, a solution is considered Pareto optimal if none of its objective function values can be improved without worsening at least one other objective. Using this concept, we can generate a set of intermediate, but "equally optimial", solutions to explicitly analyze the trade-offs between minimizing different objective functions.

Definition 2. *Given* $\boldsymbol{F}(\boldsymbol{x}^*)$, $\boldsymbol{F}(\boldsymbol{x}) \in Z$, $\boldsymbol{F}(\boldsymbol{x}^*)$ *dominates* $\boldsymbol{F}(\boldsymbol{x})$ *if* $\boldsymbol{x}^*$ *serves as a counter-example to the Pareto optimality of* $\boldsymbol{x}$.

Definition 3. $\boldsymbol{F}(\boldsymbol{x}^*) \in Z$ *is non-dominated if* $\boldsymbol{x}^*$ *is Pareto optimal. Otherwise,* $\boldsymbol{F}(\boldsymbol{x}^*)$ *is dominated.*

This domination relation is important, as it induces a partial order on the attainable set.

For our discussion, the notion of weak Pareto optimality is also helpful as it facilitates representation of the nature of the trade-off between objectives. These concepts are illustrated in figure 1.1.

Definition 4. $\boldsymbol{x}^* \in X$ *is weakly Pareto optimal* $\iff \nexists \boldsymbol{x} \in X \colon F_i(\boldsymbol{x}) \leq F_i(\boldsymbol{x}^*) \forall i \in [1,k]$.

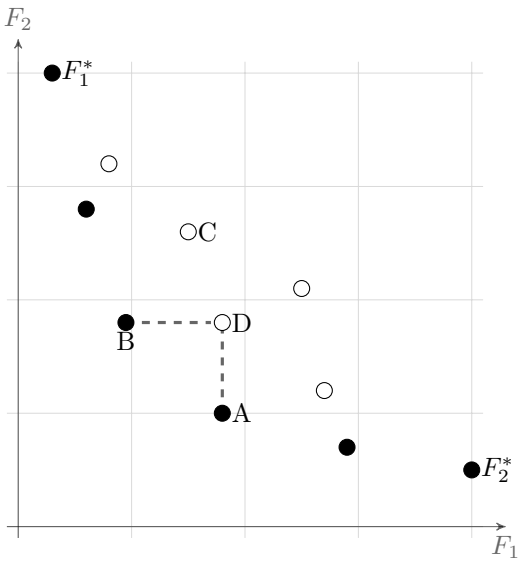

Figure 1.1: Points A and B are clearly Pareto optimal with respect to objectives F_1 and F_2 while point C is not. Point B dominates point C; however, point A does not. Point D is weakly Pareto optimal. The set of black points forms a discrete representation of the Pareto optimal front.

1.3 Pareto Fronts & Manifolds

The utility of the Pareto optimality concept is apparent when considering conflicting objectives (those that can not be simultaneously optimized) without an explicit articulation of objective preference. In the absence of this ranking, all Pareto optimal solutions can be regarded as "equally optimal" as each will offer a unique combination of objective values that can not be trivially improved.

Fully permitting the arbitrary articulation of preference, a posteriori, as well as acting upon resulting choice of solution requires knowledge of all Pareto optimal points in both design and objective space.

Definition 5. *The Pareto optimal manifold is the set of all Pareto optimal design variables,* $\{\boldsymbol{x} \in X : \boldsymbol{x}$ *is Pareto optimal w.r.t.* $\boldsymbol{F}\}$

Definition 6. *The Pareto optimal frontier, or simply Pareto front, is the set of all Pareto optimal points' objective function values. It is the image in objective space of the Pareto manifold under the* $\boldsymbol{F}$ *mapping.*

Since, at this point in the discussion, we place no substantive restrictions on the nature of $\mathbf{F(x)}$, we can say very little about general Pareto front characteristics. Complex fronts with discontinuities and collapsed dimensionality occur frequently, even for objectives and constraints with relatively simple functional forms [8].

The utility of Pareto fronts and manifolds stems from the fact that they explicitly map the optimal values that can be achieved when simultaneously considering conflicting objectives, thus permitting the user to make a more informed decision when considering design alternatives. The true Pareto front, however, can be an uncountably infinite set, which is both impractical to compute and present to the decision-maker. We, therefore, aim to capture a representative sampling of Pareto optimal points for selection by the end-user. In constructing this discrete approximation of the Pareto front, we place special emphasis on producing an evenly distributed set of solutions, as see in figure 1.2 so as to economically compute and represent the front. This avoids unnecessarily under-sampling critical regions or oversampling smooth portions of the front, both of which induce additional computational load. A uniform discrete approximation also facilitates interpolation between design alternatives.

1.4 Simulation-based Optimization

We evaluate the effectiveness of various methods for obtaining this discrete Pareto front approximation in the context of simulation-based optimization. While each subdomain has its own problem characteristics or specific tendencies, one of the transcendent qualities of simulation-aided design is computationally expensive objective functions. As a result, the key focus of an efficient optimization system lies in mitigating these costs.

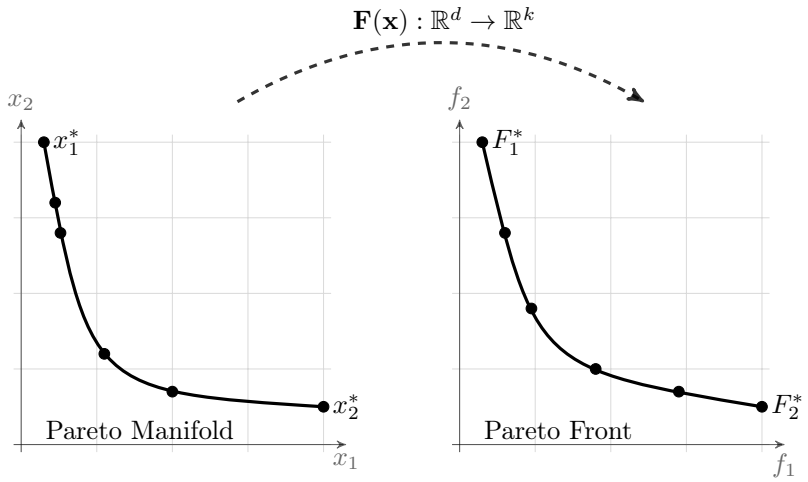

Figure 1.2: The Pareto manifold maps to the Pareto front when transformed by the vector of objective functions, $\mathbf{F}(\mathbf{x})$.

One strategy is to minimize the number of function evaluations required to reach a meaningful solution. This entails tuning the underlying scalar optimizer to the specific needs of the problem. For instance, this means employing gradient-based optimizers for smooth objective functions or using derivative-free heuristics when noise or sharp discontinuities are present. In either respect, the ability to adapt the scalar optimization strategy is desirable.

We can further ameliorate the cost of frequently running complex simulations by exploiting parallelism, both at the level of the simulation-code and the optimizer itself, as in [13]. To this end, minimizing the data dependencies at each step of the algorithm as well as favoring local and asynchronous message passing over global or synchronous communication wherever possible enhances parallel efficiency and scalability.

Lastly, adapting the objective function, itself, by replacing it with a computationally cheaper approximation, where appropriate, is an option. In practice, one can utilize lower-fidelity simulations (perhaps with coarser spacial or temporal resolution) early in the optimization process, when we are presumed to be far from the optimal values [14]. Alternatively, we can use a surrogate objective that interpolates between previously computed instances of the true objective function [30]. For a small number of design variables, it is even feasible to run a predefined set of sample simulations in an embarassingly parallel fashion, use the results to train an estimator, such as a Radial Basis Function Network (RBFN), and apply the multi-objective optimization to this much cheaper surrogate, as in [2]. This allows the user to precisely specify, in advance, the total computational expense of the problem at the cost of a boundable degree of fidelity to the true objective function.

1.5 Scalability

While we attempt to mitigate the effect of expensive function evaluations using the techniques described above, we must analyze the costs associated with various algorithmic formulations. Most importantly, we are interested in the factors that limit application to various problem types, therefore we turn our attention to parallel scalability.

We will evaluate the performance of algorithms in terms of strong scalability, in which the problem size remains fixed and the number of processors is increased, as opposed to weak scalability, where both problem size and processor count increase in fixed proportion.

Our focus involves sampling the Pareto fronts in multi-dimensional spaces; therefore, we draw attention to the 'curse of dimensionality', the exponential effect of dimension increase on sampling density. Since we aim for uniform front representation, we attempt to guarantee a particular distance between neighboring sample points. To maintain a specific separation of sample points, however, with increasing dimension entails an exponential increase in the number of required samples. Concretely, a unit length can be representated with ten samples spaced 0.1 units apart. Guaranteeing this distance when sampling a unit-square requires 100 points, while a unit-cube demands 1000 samples. Moreover, the number of other points in the neigborhood of, or lying within a given distance from, a particular sample will also increase with dimension, a fact that has important implications with respect to distributed scalability.

With this in mind, it is plain that comparing performance based on the number of sample points, S, alone makes little sense when considering problems of different dimensionality. In order to separate the concepts of problem size and problem complexity, we will evaluate scalability with respect to both sampling density per dimension, N, as well as dimensionality of the problem, k. Specifically, increasing N corresponds to a uniform increase in sampling resolution of the Pareto front, while increasing k means evaluating the effect of additional objectives on the set optimal design alternatives.

Chapter 2

Related Work

As a result of the ubiquity of problems that fit the formulation given in (1.1), a variety of methods have been devised for sampling high dimensional Pareto fronts. Here, we review some of the most popular approaches and comment on aspects like economy of function evaluations and amenability to parallel computing environments.

2.1 Scalarization Methods

As mentioned in the previous chapter, we can combine the set of objectives with a scalarization function of the type specified in (1.5). By evaluating user preferences and encoding these in the parameters of the scalarization function, we can leverage scalar optimization tools to determine an optimal solution, which, if the scalarization function meets certain conditions (see [21]), is guaranteed to be Pareto optimal. Scalarization methods leverage this fact by systematically exploring the scalarization function's parameter space. The generic blueprint is to generate a range of parameterizations, which correspond to possible user preference encodings, and subsequently solve a set of embarassingly parallel scalar optimization problems, each of which results in a Pareto optimal solution. In the end, this collection of solutions and corresponding objective values constitute a discrete approximation of the Pareto manifold and front, respectively.

This procedure can be concretely illustrated as follows. Consider a basic scalarization function, like the weighted sum method

$$s(\mathbf{F}) = \sum_{i=1}^{k} w_i F_i(\mathbf{x}) \text{ with } w_i \geq 0 \ \forall i \in [1, k] \tag{2.1}$$

where all objectives are assigned a weight, w_i, and combined linearly. By stepping through the $(k-1)$ dimensional normalized weight space and solving problem (1.6), one can generate a set of Pareto optimal points.

While the approach is straightforward, the mapping between weight space and location on the Pareto front generally is not and fronts generated this way

suffer from highly irregular sampling. Making matters worse, this mapping is not even guaranteed to be surjective, meaning certain portions of the Pareto front may not be "captured" by this method at all [21].

These shortcomings are addressed in a number of different ways. Notably [3] proposes a exponential weighting scheme that enables the capture of Pareto points in non-convex regions at the cost, however, of additional tuning parameters and without significantly improving the sampling irregularity. Another interesting approach described in [11] involves restricting the minimization domain to a ray emanating from a particular point outside the attainable set. While this provides a simple formulation and maintains some of the appealing properties of this class of methods, its generalization to multiple dimensions is non-trivial and seems to produce equally spaced front representations under specific projections.

In general, scalarization methods provide a simple way to generate points in the Pareto optimal set. Moreover, since they typically involve solving a predefined set of scalar optimization subproblems, they are amenable to parallelization on a massive scale. While the lack of information exchange among the subproblems allows them to scale well, both with problem size and complexity, the sampling quality often suffers, and these methods are not well suited to highly non-linear problems.

2.2 Population-based Methods

Rather than attempting to break the overall sampling task into a series of subproblems, as in the proposed scalarization approach, population-based methods, generally speaking, attempt to solve the problem as a whole. This is done, typically, by generating sets of mutually non-dominated solutions.

A number of different heuristics have been proposed for generating and maintaining these populations using, for instance, particle swarm optimization and scatter search [25] or simulated-annealing [31]. However, the preferred methods remain Evolutionary Algorithms (EAs), which draw on biological analogy to generate random "individuals" that represent potential solutions and apply the principle of natural selection by preferrentially recombining elements of solution candidates with better "fitness" to produce new offspring. The central idea is that, by iterating this procedure for a number of generations, we evolve a population of mutually non-dominating individuals that approximate the Pareto manifold. A number of specific algorithms have been proposed (see [33], [21], [7], and [26]) using a variety of techniques to balance the discovery of non-dominated solutions with diversity criteria to discourage solution crowding.

EAs robustness and ease of application makes them the de-facto method for solving complex multi-objective optimization problems. By randomly sampling and recombining elements of successful design variable sets, they require no information about the objective function gradients, nor do they even require that one exist. Moreover, because of the opportunisitic sampling behavior, these algorithms are robust against failures of function evaluations. For instance, if

a requested solution returns invalid results (as a result of simulation failure or poor numerical accuracy), the algorithm simply carries on evaluating other candidates. Finally, a mature infrastructure that modularizes the generation and selection of non-dominated solutions [4] facilitates experiementation with multiple heuristics to determine the best approach to solving a particular problem.

While EAs are capable of performing well on even the most difficult problems, some key aspects suggest that they are not ideal for all objective types. First, as in [7], solutions are evaluated based on their relative dominance to other solutions in the population. Unlike scalarization methods that provide some necessary or sufficient conditions for optimality of solutions, this does not actually guarantee Pareto optimality of the resulting sample points. As a result, the output may not accurately represent the true Pareto front. Secondly, because EAs typically shun gradient information in favor of empirical exploration of the solution space, they can be less efficient (requiring more costly function evaluations) in situations involving smooth or convex objective functions or where derivative information is readily available or easily obtainable.

Finally, the nature of the non-dominating fitness criteria inherently limit the scalability of this class of algorithms. Since this criterion, by definition, is relative, it involves comparing elements of the population against each other, which naturally implies some super-linear running time behavior, scaling with front total sampling density as $\mathrm{O}\left(S^2\right)$ according to [7] and reduced to $\mathrm{O}\left(S \log_{k-1}(S)\right)$ in [16] with some clever partitioning. Parallelizing the evaluation [10] or distributing various objective space regions to different processors [9] can mitigate this effect; however, super-linear run-time scaling remains. The effect is particularly stark when increasing the number of objectives, k, which requires not only a corresponding exponential increase in population size (to maintain an evenly sampled Pareto front) but also distorts the utility of the dominance ranking criterion. This leads to poor scalability of EAs as applied to high-dimensional problems; the hypersurface-area to hypervolume effect permits a higher proportion of non-dominated solutions and weakens selection pressure [15].

2.3 Geometric Methods

Rather than rely on a flood of function evaluations to explore the solution space, a final class of methods uses the scalarization approach, but exploits geometrical arguments to further restrict the attainable set for each optimization subproblem. Practically, this is accomplished by adding auxilary constraint functions to steer the scalar optimization subroutines towards desirable solutions and deliver an evenly sampled front. The Normal Boundary Intersection [6] and the Normal Constraint [22] method essentially restrict each optimization subproblem to only consider solutions that lie on the normal vector emanating from the convex hull of the individual minimizers (CHIM).

These methods make intuitive sense, as well as produce well sampled fronts. However, by restricting the attainable set to the CHIM normal, these approaches are only capable of capturing Pareto points that lie within the projection of this

hull. Furthermore, these points are only equally distributed when viewed from the CHIM frame, as seen in figure 2.1.

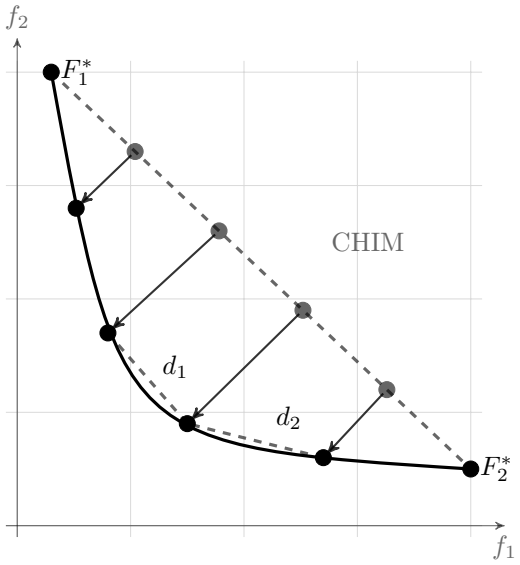

Figure 2.1: In the NBI and NC methods, evenly spaced points on the CHIM are projected to the Pareto front. Unfortunately, as a result of the front geometry, the resulting sample points are not always evenly spaced. The fact that $d_1 \neq d_2$ illustrates this effect.

The specific algorithms also exhibit some scalability issues, especially when increasing the number of objectives, k. Since the boundaries of the Pareto front are not know a priori, it is often difficult to decide where to place solution points on the CHIM. In [23] and [24], it is suggested to divide the solution algorithm into two phases; the first samples the Pareto fronts of all combinations of individual objectives to establish the boundaries of the complete front. The second phase involves solving a set of scalar optimization subproblems to find Pareto optimal points on the interior of this boundary. The first phase, consists of $\sum_{i=2}^{K-1} \binom{K-1}{i}$ "sub-fronts" to sample before the domain is properly bounded and sampling on the interior can begin. Furthermore, the recursive nature of sub-front sampling, such as this, limits the scalability to only i parallel sub-front's per step of the initial phase.

Chapter 3

Homotopy Methods

While some common solution methods introduce additional mechanisms to encourage solution diversity and spread, continuation and homotopic methods attempt to build evenly spaced Pareto front representations by construction. Like Normal Boundary Intersection (NBI) and Normal Constraint (NC) approaches (see section 2.3), this is typically accomplished by augmenting the problem constraints, $\mathbf{h}(x)$, to reduce the feasible solution space to a more desirable subset of the whole objective space. The precise manner in which these constraints are formulated, however, has important implications for the scalability of the solution method as a whole. In this section, we investigate various constraint formulations, as well as examine their performance.

3.1 Continuation Method

A straightforward method for constructing a well-sampled discrete Pareto front representation, as discussed in [27] and [28], involves adding explicit solution spacing terms to the equality constraint set. For instance, consider a standard bi-objective minimization problem.

$$\begin{array}{c} \min_{\mathbf{x}\in X} \mathbf{F}(\mathbf{x}) = [F_1(\mathbf{x}), F_2(\mathbf{x})]^T \\ \text{subject to} \\ g_j(\mathbf{x}) \leq 0 \quad \forall j \in [1, m] \\ h_l(\mathbf{x}) = 0 \quad \forall l \in [1, e] \end{array} \tag{3.1}$$

Using a simple weighted sum scalarization, we obtain the following scalar objective function,

$$f(\mathbf{x}, \lambda) = (1 - \lambda)F_1(\mathbf{x}) + \lambda F_2(\mathbf{x}) \text{ with } \quad 0 \leq \lambda \leq 1 \tag{3.2}$$

the minimizer of which, $\min_{\mathbf{x}\in X} f(x, \lambda) = \mathbf{f}(\lambda)$, is a parametric expression of the Pareto front. The endpoints, $\mathbf{f}(0)$ and $\mathbf{f}(1)$ correspond to the minima of the individual objective functions, $F_1(\mathbf{x})$ and $F_2(\mathbf{x})$, respectively, while the parameter λ controls the transition between these extremes.

As mentioned before, the mapping between λ and position along the Pareto front is often highly nonlinear. However, as presented in [28], we can exploit this additional degree of freedom to sample specific points along the curve by treating the λ parameter as a design variable. This added flexibility is complemented by an auxilary equality constraint, typically of the form

$$||\mathbf{F}(\mathbf{x}) - \mathbf{F}_{prev}||^2 = \gamma^2 \tag{3.3}$$

that explicitly enforces equal spacing between adjacent sample points. This allows a suitable scalar optimization procedure to minimize the scalarized objective function (3.2) subject to the constraint that the result lies a certain distance, γ, from a specified point, $\mathbf{F}_{prev}$. By seeding this procedure with either of the objective minima, for instance with $\mathbf{F}(\mathbf{x}_0) = \mathbf{f}(0)$, we can iteratively generate evenly spaced samples by "marching" along the Pareto front carrying out the following scalarized minimization

$$\begin{array}{c} \min \quad (1-\lambda_n)F_1(\mathbf{x}_n) + \lambda_n F_2(\mathbf{x}_n) \\ \mathbf{x}_n \in X \\ 0 \leq \lambda_n \leq 1 \end{array} \tag{3.4}$$

subject to

$$||\mathbf{F}(\mathbf{x}_n) - \mathbf{F}(\mathbf{x}_{n-1})||^2 = \gamma^2 \tag{3.5}$$

for n up to the number of desired sample points, N (provided a suitable step-size, γ).

In [27], the authors demonstrate application of this method to a variety of simple bi-objective test problems from the literature and favorably compare the results to those obtained obtained using EAs (specifically NSGA-II). We confirmed these results for a subset of these problems, specifically

$$\min \begin{bmatrix} (x_0-1)^4 + (x_1-1)^2 + (x_2-1)^2 \\ (x_0+1)^2 + (x_1+1)^4 + (x_2+1)^2 \end{bmatrix} \tag{3.6}$$

and

$$\min \begin{bmatrix} 1 - \exp(-\sum_{i=1}^{3}(x_i - 1/\sqrt{3})^2) \\ 1 - \exp(-\sum_{i=1}^{3}(x_i + 1/\sqrt{3})^2) \end{bmatrix} \tag{3.7}$$

Here, figures 3.1 and 3.2 compare the continuation results to naive weighted sum scalarization methods. For comparison, we provide the true Pareto front as well as results obtained via NSGA-II in figure 3.3.

While this method produces well-sampled Pareto front representations, even for difficult problems, it requires sequential variation of the λ-parameter and serial computation of the sample point locations, thereby limiting its scalability and application to large problems and those with expensive objective functions.

3.2 Parallel Front Computation

The authors address this concern in [28] by extending the algorithm to parallel processing environments. This is done, primarily, by replacing references to

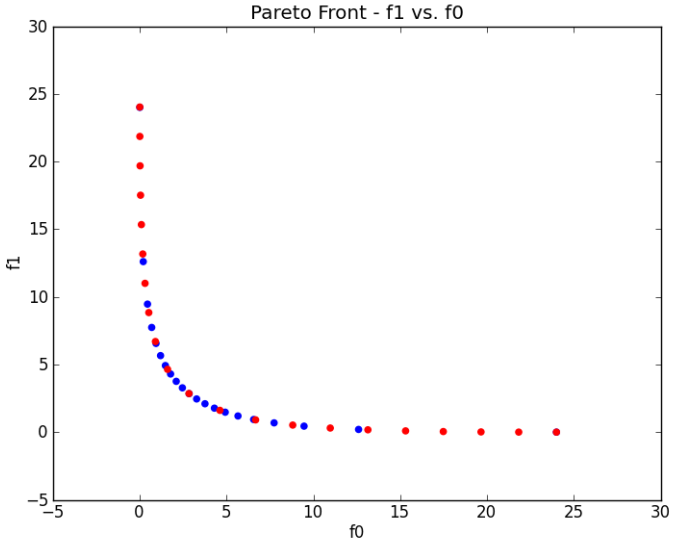

Figure 3.1: The Pareto front for the bi-objective problem (3.6). The results from the the classical weighted sum method are shown in blue, while the continuation method results using the algorithm from [27] are shown in red.

previously computed sample points with continually refined position estimates. This way, we create a set of semi-independent and simultaneously-solvable scalar optimization sub-problems, coupled only through the auxilary equispacing constraint (3.3). In fact, it can be easily seen that we can distribute the subproblems to as many processors as we have Pareto front sample points, thereby lifting the scalability limits imposed by the sequential algorithm.

This parallel approach, however, does require a non-obvious initialization step, where the various $\mathbf{f}(\mathbf{x}_n)$ values are seeded with a set of initial position estimates, or ansatz front. Since the authors of [28] only apply this method to bi-objective problems, generating these initial estimates is rather straightforward; the ansatz front is simply a set of N equally spaced points linearly intepolated between the individual function minima, $\mathbf{F}(\mathbf{x}_1^*)$ and $\mathbf{F}(\mathbf{x}_2^*)$.

With the initial estimate in place, and the parallel refinement procedure mentioned above, rather than sequentially "marching" along a curve (as in section 3.1), we interatively "bend", or homotopically deform, the ansatz into an equispaced discrete Pareto front representation. This process is illustrated in figure 3.4.

One concern with enforcing an explicit spacing constraint like (3.5) involves selecting a suitable step size parameter, γ. The authors recommend using

$$\gamma = \alpha \frac{||\mathbf{F}(\mathbf{x}_1^*) - \mathbf{F}(\mathbf{x}_2^*)||}{N} \tag{3.8}$$

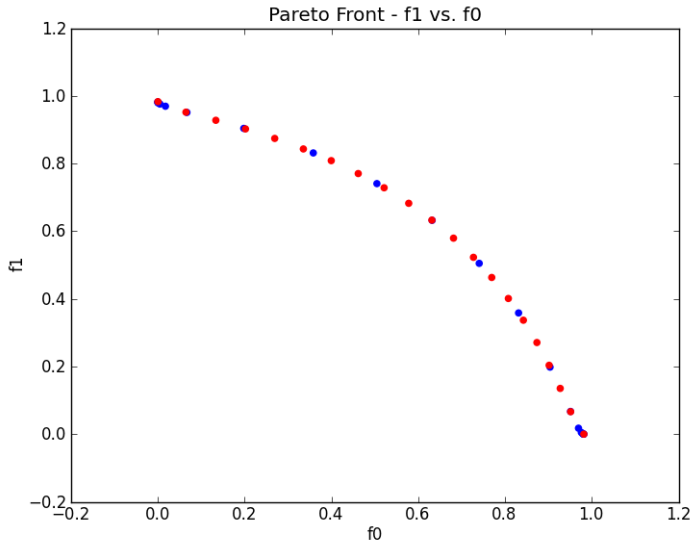

Figure 3.2: The Pareto front for the bi-objective problem (3.7). The results from the the classical weighted sum method are shown in blue, while the continuation method results using the algorithm from [27] are shown in red.

where the α factor is designed to compensate for the arc-length of the actual Pareto front being larger than that of a straight line connecting the images of the individual objective minimizers. The problem is that, without additional information about the length of the true Pareto front, this parameter can not be established a priori. Any error in setting this parameter can cause uneven sampling of the final front, by (in the case of underestimating α) neglecting the tail of the front, or wasted computational effort through the inclusion of Pareto dominated solutions, as an overly zealous α pushes the sampling procedure past the $\mathbf{F}(\mathbf{x}_2^*)$ endpoint. One can imagine a bootstrapping process whereby the total arc-length of the front is estimated and refined by successive applications of this algorithm; however, this would involve serial applications of the front sampling procedure, again limiting scalability. Finally, this issue gets more tenuous with an increase in the number of objectives, as a proper α must be established for each of the $k-1$ dimensions of the Pareto front.

3.3 An Alternative Constraint Formulation

A simpler approach that accomplishes the same goal is to adjust constraint (3.5) to, instead, explicitly enforce equal spacing between a point's up- and down-stream neighbors. Specifically, a modified constraint of the form

$$||\mathbf{F}_n - \mathbf{F}_{n+1}||^2 - ||\mathbf{F}_n - \mathbf{F}_{n-1}||^2 = 0 \tag{3.9}$$

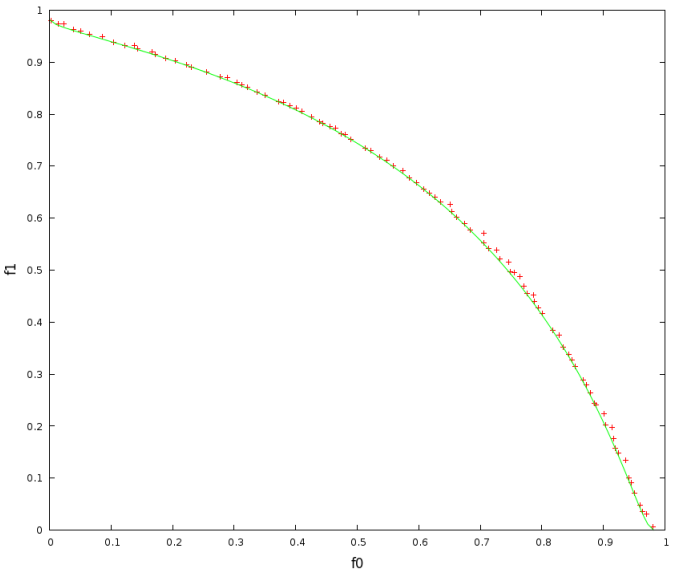

Figure 3.3: A discrete approximation to the Pareto front for the bi-objective problem (3.7) produced by NSGA-II is shown in red, while the true Pareto front is displayed in green.

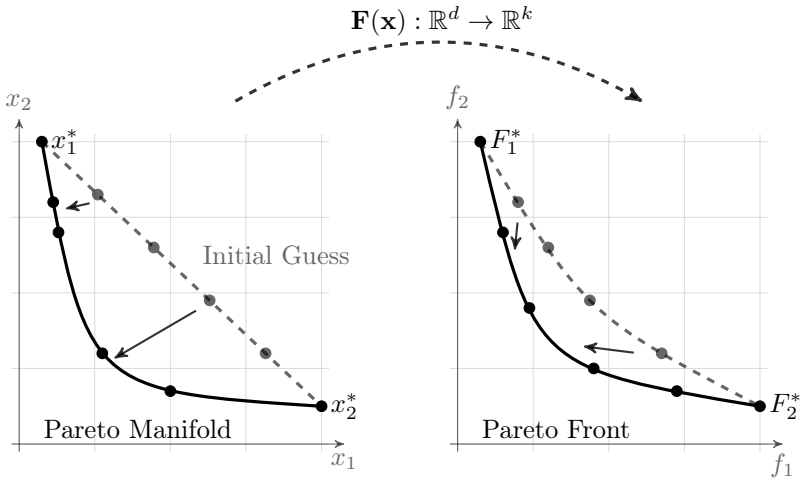

Figure 3.4: The linearly interpolated ansatz is homotopically deformed into a suitable discrete Pareto front representation by iterating the parallel procedure defined by (3.4) and (3.5).

eliminates the need to estimate the Pareto front arc-length a priori by not fixing the spacing between points and allowing the front representation to expand

indefinitely.

Rephrasing the constraint this way, however, has other important ramifications, as well. First, it ameliorates an unfortunate consequence of the formulation in (3.5) that limits the optimizer's improvement per iteration to γ. Because of this limited step size, if the ansatz is sufficiently distant from the real Pareto optimal front, applying the method of [28] will require multiple iterations to generate a set of truly optimal solutions. Constraint (3.9), however, is more in line with Normal Boundary Intersection and Normal Constraint methods (described in section 2.3) that do not restrict step sizes, allowing the sample points to converge to the Pareto front more quickly. For a whole series of problems, in fact, all sample points reach the optimal front on the first iteration, with subsequent iterations serving only to improve sample distribution along the front.

Most importantly, by eliminating explicit references to the total arc-length, only local communication of current $\mathbf{F}_n$ values between adjacent optimization sub-problems is required. This means we can concurrently compute uniformly distributed Pareto front representations using only local, asynchronous communication between neighboring compute nodes, a key feature of massively scalable algorithms. In fact, the amount of parallelism is bounded only by the number of Pareto front sample points required, N, which (to obtain a truly uniformly spaced representation) often scales exponentially with the number of objectives.

Another convenient aspect of the alternative constraint formulation (3.9) is its amenability to multi-objective generalization. For each additonal objective, we need only to add another λ scalarization parameter to (3.2) as well as another dimension to the ansatz. This, in turn requires an additonal constraint, analogous to (3.9), to guarantee equal spacing of the sample points. As discussed above, this allows the ansatz front to expand as necessary in each dimension and preserves the locality of the algorithm by relying only on data from the immediate neighbors. Furthermore, allows us to orthogonalize the warping of the ansatz along mesh-defined axes.

Listing 3.1: Proposed Algorithm

1. Generate an ansatz front or initial set of $\mathbf{F}_n \forall n \in [1, N]$
2. Loop while $\mathbf{F}_n$ is unsatisfactory
3. Loop over $n \in [1, N]$
4. Augment the inequality constraints $\mathbf{g}(\mathbf{x}) \leq 0$ with $0 \leq \lambda_i \leq 1 \forall i \in [1, k]$
5. Augment the equality constraints $\mathbf{h}(\mathbf{x}) = 0$ with

$$||\mathbf{F}(\mathbf{x}_n) - \mathbf{F}_{n+1}||^2 - ||\mathbf{F}(\mathbf{x}_n) - \mathbf{F}_{n-1}||^2 = 0$$

   for each axis of the ansatz
6. Solve $\min_{\mathbf{x}_n \in X, \lambda} \lambda_1 F_1(\mathbf{x}) + \lambda_2 F_2(\mathbf{x}) + \ldots + \lambda_k F_k(\mathbf{x})$ subject to the above augmented constraints
7. Store new $\mathbf{F}(\mathbf{x}_n)$ values
8. End loop 3
9. End loop 2

A general overview of the proposed algorithm, precisely defining the additional constraints and solution procedure.

3.4 Ansatz Generalization

Generalizing this approach to k dimensions requires not only auxiliary constraints and scalarization variables, but a full generalization of the ansatz concept as well. In [28], the authors use a set of equally spaced points lying on the line connecting the individual objective minimizers. This initial mesh is then homotopically transformed into a curve representing the Pareto front by applying the optimization operator to each of the constituent points. In seeking to extend this ansatz concept to arbitrary dimensionality, we identified a number of vital requirements for any meshing solution.

1. An ansatz mesh must have an approximately equal distribution of points over the $k-1$ dimensional convex hull of the k individual function minimizers. If this is not the case, this will result in either uneven sampling of the final Pareto front (a consequence we are attempting to avoid by design) or require many additional iterations of the optimizer to propagate the solutions throughout the resulting Pareto front.

2. Any meshing solution should easily support a range of sampling resolutions, allowing the user to explicitly specify the level of detail required.

3. The ansatz solution should scale with problem dimension. Concretely, the alternative constraint mentioned above introduces a data dependency to the optimization process by requiring the location of each point's neighbors. As a result, the ansatz solution should minimize the number of additional neighbors required by each point as the dimension of the mesh increases.

4. Since this method is intended for massively parallel computing architectures, the meshing solution should support distributed generation with low computational, as well as storage, overhead.

To address these requirements, we supply an ansatz with the simplicity and scalability of a rectilinear, coordinate parallel mesh that is contained within the convex hull of the individual objective minimizers. This is done, first, by remapping the k function minimizers to the corners of a unit $k-1$-simplex. We address requirement 2 by selecting a sampling density that is taken as equal for all axes. From this, we generate the interior and boundary points of the mesh by forming the set of $k-1$-tuples that lie on or under the hyperplane connecting the non-zero corners of the $k-1$-simplex. This means finding all coordinate tuples, $\mathbf{c}$, satisfying the following relation.

$$\sum_{i=1}^{N} c_i \leq N \tag{3.10}$$

This set of $k-1$-tuples makes it relatively straightforward to determine a point's neighbors by simply incrementing or decrementing each coordinate. Moreover, the coordinate-parallel nature of the tensor-product mesh addresses characteristic 3 by adding only two more mesh point neighbors per additional dimension of the ansatz. In other words, the degree of each node in the graph is bounded by $2(k-1)$. Ultimately, this means that the number of additional constraints added the overarching optimization problem to ensure equispacing of the Pareto front, as well as the magnitude of the data dependency of the optimization sub-problem occurring at each mesh node, scales linearly with the number of target objectives. This has important implications for parallel scalability of the overall method, as addressed in section 4.3. Next, these $k-1$-tuples are remapped to the original objective space through a straightforward barycentric coordinate transform that addresses requirement 1 by preserving the coordinate-wise equal spacing between the points comprising the ansatz solution. This renders the ansatz a uniformly-distributed linear interpolation between the k individual objective minimizers, as seen in figure 3.5.

Finally, since all of the coordinate transforms and neighbor-discovering operations can be accomplished independently for each point, this meshing strategy supports fully distributed generation, satisfying requirement 4. The methods

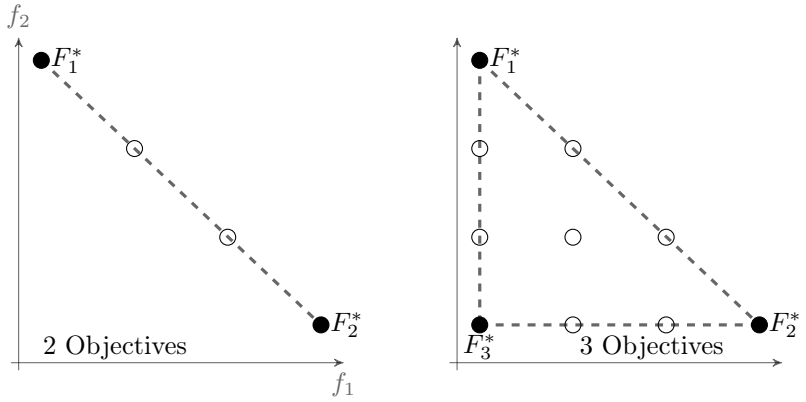

Figure 3.5: Bi-objective and tri-objective general ansatz examples with four points per axis. In the tri-objective case at right, the points are projected into the $f1 - f2$ plane for easier viewing.

involved do require a k-level recursive function call and k separate integer bisection searches (on sets whose size scale with the per-axis sampling resolution), yielding linear complexity (in k). However, these mappings are computed only during mesh instantiation in the set-up phase with the results stored (implicitly or otherwise) in the mesh data-structure; moreover, while the approach is general, in all the applications presented here, k is kept to single digits.

3.5 Optimizer Selection

Having cast (1.1) as a series of scalar-valued sub-problems, [28] somewhat arbitrarily employs a closed Sequential Quadratic Programming (SQP) code to perform the individual sub-problem optimizations. We, however, wished to investigate the performance of different algorithms and leverage freely available scalar optimization software. To this end, we selected the NLopt package [1] since it provides a general and uniform C++ interface to a variety of optimized implementations.

The primary difficulty, however, involves the addition of constraint (3.9), which is both non-linear and non-convex. Therefore, in order to experiment with the full range of solvers, we employed the augmented Lagrangian approach, which effectively adds violated constraints to the objective function using a penalty term [5]. We attempted to solve the scalarized sub-problems using a variety of stochastic and derministic, as well as derivative-free and gradient-based, algorithms. This indirect method of addressing the problem constraints, however, produced unsatisfactory results, often requiring thousands of function evaluations per sub-problem.

[1] The NLopt nonlinear-optimization package developed by Steven Johnson, see http://ab-initio.mit.edu/nlopt

We then turned our attention to algorithms that could directly incorporate both equality and inequality constraints. These included the COBYLA algorithm [29] (Constrained Optimization By Linear Approximation), which constructs linear approximations to the objective and constraint functions and progresses towards the feasible minimum using a trust region approach. This method was often able to solve simple test problems with only a few hundred objective function evaluations per sample point.

Unsurprisingly, however, the SLSQP algorithm [17] (Sequential Least-Squares Quadratic Programming), which similarly minimizes quadratic approximations to the objective and constraint functions, outperformed COBYLA by only requiring a few dozen function evaluations per sample point. SLSQP does require gradient information, supplied by either simple forward or central finite difference methods in our implementation, to construct the set of quadratic programs, which does increase the number of function evaluations per iteration. However, the better objective approximation scheme allows the algorithm to take larger steps, reducing the total number of iterations by an order of magnitude over COBYLA. Futhermore, in cases of objective functions with readily available gradient information or differentiable function surrogates, the multiplicative effect of the finite differences approximations disappears.

Chapter 4

Parallelization

With the general algorithm in place, we turn our attention to efficiently computing uniformly spaced discrete Pareto front representations in a parallel environment. Given that our target application domain involves particle accelerators, we leverage the Opt-Pilot framework, a multi-objective optimization system for large-scale simulation-based problems (originally developed in the context of beam dynamics studies) described in [12] and [14]. We describe the design and implementation of a homotopic optimizer submodule as well as the domain paritioning scheme used to distribute the front sampling to multiple processing elements with minimal synchronization.

4.1 The Opt-Pilot Framework

Originally designed to tackle multi-objective optimization problems in the context of particle accelerator physics, the Opt-Pilot framework provides the necessary set of well-defined interfaces and interconnects to solve simulation-based problems with modern HPC hardware. Key to the framework's efficacy is the event driven architecture, division of labor between various component processes, agnosticism to simulation (as well as optimization) implementations, and flexible process mapping.

As an event driven system, all of Opt-Pilot's main components descend from a basic Poller class, which, upon instantiation, runs an infinite message dispatch loop that directs asynchronously received MPI messages to an appropriate handler. This abstract class also supplies hooks such that refining implementations can define the precise behavior of the component. The most important consequence of this aspect is the event-driven nature of the system, facilitating asynchronous message exchange, minimizing synchronization, and isolating state.

Opt-Pilot presents three primary abstractions for dividing responsibility among process instances. Firstly, and most relevant, the Optimizer abstraction is responsible for deciding which sets of design variables to evaluate and

directing the optimization based on the responses obtained. As it refines the Poller class, it implements the interface defined in listing 4.1, often with the bulk of the work being done in the "onMessage" handling routine and the "postPoll" routine (for updating the Optimizer state).

Listing 4.1: The public interface implemented by a refinement of the Optimizer class in the Opt-Pilot framework.

```
{
        virtual void initialize () = 0;
        // Polling hooks
        virtual void setupPoll () = 0;
        virtual void prePoll () = 0;
        virtual void postPoll () = 0;
        virtual void onStop () = 0;
        virtual bool onMessage( MPI_Status status,
                                size_t length) = 0;
}
```

Secondly, the Worker class is responsible for mapping the input design variables to an ordered set of objective values. This mapping can be arbitrarily complex and, as the system is designed for large-scale simulation-based problems, is able to involve additional processes (referred to as Co-Workers) to speed up a parallelizable simulations.

Finally, the Pilot class handles the coordination between these two components, queuing and routing evaluation requests to idle Worker instances and returning simulation results to the Optimizers that require them. This decouples the two layers, presenting the set of Worker processes as a coherent evaluation service and, if necessary, coordinating solution exchange among Optimizers through a managed topology.

With the Pilot instances mediating communication between Optimizers and Workers, the specific implementations of both are freely able to change based on the needs of the problem at hand. At the moment, the Opt-Pilot package supports EA-(implementing the PISA interface [4]) and homotopy-based optimizers, as well as forward solvers with varying time complexity and resolution (see [1] and [12]). Besides facilitiating rapid and effective Pareto front sampling for a variety of problems, this provides a unique environment to directly compare optimization strategies in a massively parallel system.

4.2 Homotopy Optimizer Implementation

In order to obtain maximum parallelism and fully leverage the scalability of the framework, we need to mirror the decoupling between requesting an evaluation and processing the result. EAs are well suited to this regime since the data dependency between organisms arises only during the variator step (spawning new generations), which is designed to be computationally cheap. Thus, gen-

erally speaking, entire generations can be evaluated in parallel. Unfortunately, the sequential nature of the SQP method for handling the optimization sub-problems requires a set of design variables to be mapped to objective values (the expensive evalution accomplished by the Worker process) before a decision is made about where next to probe the design space. Therefore, there exists a strict data dependency between subsequent SQP iterations, as seen figure 4.1.

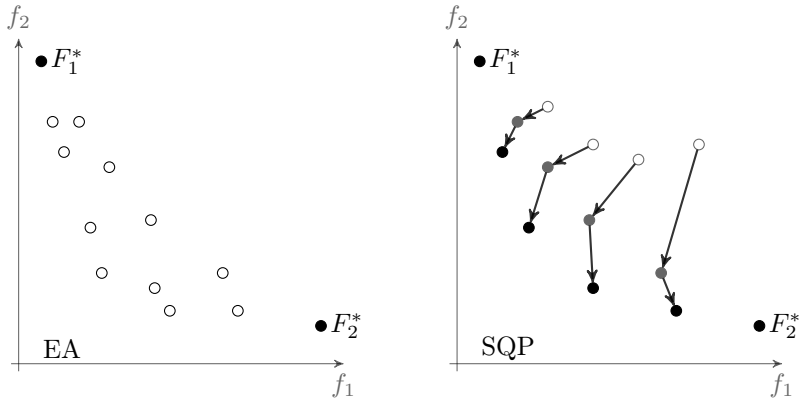

Figure 4.1: EAs spawn solutions that can be evaluated in parallel as seen on the left. The strict data dependencies in the SQP algorithm, as each iteration depends on the results from the last, are illustrated on the right.

We can, however, exploit the horizontal parallelism that exists in running sets of loosely coupled optimization sub-problems, as dictated by listing 3.1. This entails initiating a burst of N (or more, if numerical derivatives are required) function evaluations at the outset. As results are returned, the appropriate SQP instance solves its local quadratic program and requests another evalution from the Pilot before suspending itself. This model permits parallelism by multiplexing N simultaneously running SQP instances and processing iterations opportunistically (or as results become available).

One particularly limiting drawback of the NLopt API, however, is its inability to suspend the optimization process by storing internal state. Instead, the library is intended to be run to completion from a supplied starting point. Wanting to maintain NLopt's ability to easily swap out optimization algorithms (as described in section 3.5), we added an Evaluator abstraction between the SQP method and the Optimizer's request dispatch method. This serves as a mediation layer (internal to the Optimizer) that caches results as they are returned by the Pilot from the Workers.

As each SQP instance attempts to minimize its scalarized objective function, it requires mapping specific combinations of design variables to function values. The Evaluator catches these requests (as a set of design variable) and attempts to return the appropriate objective function values to the SQP instance, if they are located in the cache. If not, the Evaluator generates an evaluation request,

which is sent to the Pilot, and throws a forced stop exception, terminating the SQP routine. Once a Worker instance has successfully mapped the requested design variables to objective space, the results are returned to the Optimizer and cached in the Evaluator. At this point, the appropriate SQP instance is notified and restarted from the original starting point, but, this time, when the set of design variables in question are passed to the Evaluator, it responds immediately with the desired mapping.

4.3 Domain Decomposition

To fully employ the capabilities of modern parallel computers in applying this method to large-scale problems, we pursued a method of distributing the optimization sub-problems across a set of compute nodes. When evaluating a domain decomposition technique it is important to consider the quality of the partitioning created and the computational overhead involved in establishing the subdomains.

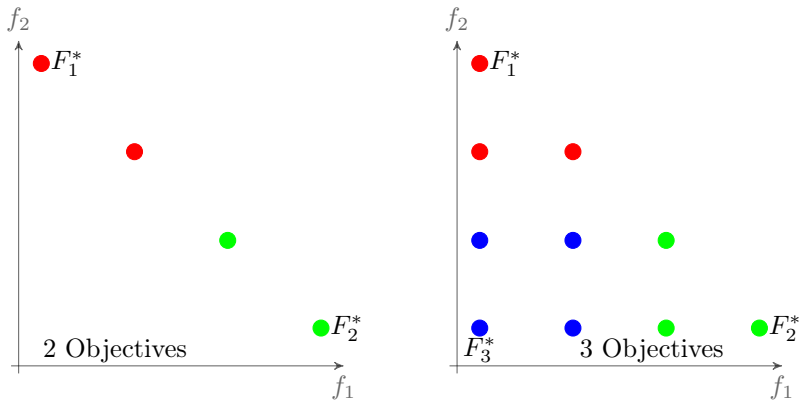

Figure 4.2: Bi-objective and tri-objective examples of domain decomposition for four points per axis and two subsets per axis. Points of the same color are handled by the same processor. This scheme increases data locality and ensures all communication occurs only between neighboring processors.

With regard to the quality of the resulting partition, we aim to minimize the amount of inter-processor communication and balance the computational load across available resources. The local nature of the algorithm requires that processes exchange and maintain the values associated with sample points on the boundary of the sub-domains. This means the communication requirements scale directly with the size of the interfaces between various groups of sub-problems. Therefore, we aim to maximize the ratio of mesh subset hyper-volume (representing the amount of work done by a compute node) to surface hyper-area (dictating the amount of data sharing and costly inter-process communication).

We selected a decomposition scheme that groups optimization sub-problems based on regular, coordinate-parallel partitions, dividing the domain into a set of hypercubes of approximately equal size. This allows for significant data re-use by grouping coupled (through the equispacing constraint (3.9)) sub-problems together and preserving data locality, as seen in figure 4.2.

The resulting collection of grouped sub-problems, or mesh subsets, are topologically similar to the original ansatz connectivity graph. This permits leveraging our earlier work on creating an ansatz structure that supports distributed mesh generation and facilitates code reuse (by sharing a number of functions between the ansatz generator and mesh decomposition classes). Most importantly, however, the distributed ansatz generator and simplicity induced by coordinate-parallel partitions allow the entire mesh to be built in parallel with absolutely no inter-process communication required.

Chapter 5

Results

5.1 Metrics

Having detailed the proposed algorithm for sampling general Pareto fronts, we turn to evaluating the performance relative to the other methods described. We seek, here, to establish a set of metrics for measuring front accuracy as well as quality (with a focus on evenly sampled representations). We compute these metrics for a set of benchmark multi-objective optimization problems taken from the literature.

5.1.1 Hypervolume

A useful and, more importantly, scalable metric for measuring the accuracy and quality of a Pareto front approximation is the hypervolume of the dominated space. Effectively, given a point in the attainable set, we measure the k dimensional volume of the dominated space, relative to some "worst-case" point, as shown in figure 5.1.

The particular convenience of this metric lies in the fact that the limit of a perfectly sampled Pareto front will converge to a concrete value, representing the total dominated volume. Given the fact that the true Pareto optimal front is the maximal set of the attainable region poset, with this metric, we are able to directly measure convergence to the true Pareto front. We link to the implementation provided by [32] for efficient computation of hypervolumes.

5.1.2 Evenness

Beyond simply converging to the Pareto optimal front, we aim to ensure a uniformly sampled approximation in an effort to maximize both economy of function evaluations as well as permit useful interpretation of results. A number of methods have been proposed to capture the essence of uniformity, from normalized root mean square (RMS) distances between adjacent points [27] to

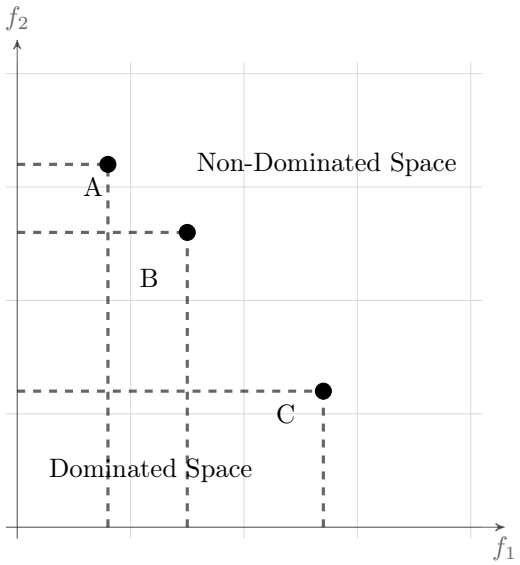

Figure 5.1: The hypervolume of a Pareto front is the total volume of the dominated space. Computed for the illustrated points with respect to the origin, this equals the volume of the union of sets A, B, and C.

the maximum lower bound of all inter-point distances [19]; however, these methods do not generalize to varying objective scales and are not well adopted by the community at large. In [22], the authors recommend a measure of "evenness" that corresponds to the intuitive notion that no region of the Pareto front is over- or under-represented by the discrete approximation. Given a point F_i in the approximation set, we first consider the nearest neighbor distance, d_i^l, to another point in the set, as in figure 5.2. Next, we consider the largest sphere, containing no points from the set, that can be constructed such that F_i and another point, F_j, both lie on the surface and denote the diameter, d_i^u. Then, given the set

$$d\colon = \left\{ d_i^l, d_i^u \colon \forall i \in [1, S] \right\} \tag{5.1}$$

we define the evenness, $\mathbf{E}$, as

$$\mathbf{E} = \sigma_d / \hat{d} \tag{5.2}$$

the ratio of this set's standard deviation, σ_d, to its mean, $\hat{d}$, implying that a perfectly uniform distribtion has $\mathbf{E} = 0$. This metric generalizes to multiple dimensions quite easily by considering hyperspheres when determining the d^l and d^u values.

5.1.3 Function Evaluations & Time-to-solution

Finally, while the previous two metrics quantify front accuracy and quality, we aim to measure the amount of computational effort required to obtain the

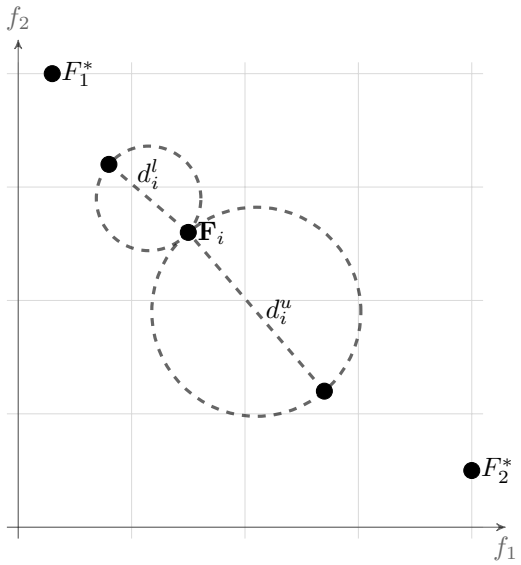

Figure 5.2: Construction of d_i^l and d_i^u for a point $\mathbf{F}_i$.

given solutions. To directly compare algorithms, independent of specific implementation or computing platform, we report the number of objective function evaluations per Pareto point used to generate front approximations for the synthetic test problems. Aiming also to demonstrate the benefits of paralleization, we report total time-to-solution in seconds, as well as examine both strong and weak scalability to measure the specific efficiency of our implementation for actual applications.

5.2 Test Problems

Since the homotopy method presented here primarily relies on augmenting the constraint functions, $\mathbf{h}(\mathbf{x})$, with additional equispacing constraints, we chose to directly compare the accuracy and quality of the Pareto front approximation with those produced by similar methods, like NBI and NC. Therefore, we selected a set of test problems taken from the recent literature [23] and where some of the above metrics are reported for a variety of scalarization and geometric methods.

5.2.1 Problem 1

The first problem is relatively easy, consisting of linear functions and convex constraints.

$$\begin{gathered}\min_{\mathbf{x}} f_i(\mathbf{x}) = x_i \text{ for } i = 1, 2, 3 \\ \text{subject to} \\ -x_1 + x_2^{-1} + x_3^{-1} \leq 0 \\ x_1^{-1} - x_2 + x_3^{-1} \leq 0 \\ x_1^{-1} + x_2^{-1} - x_3 \leq 0 \\ 0.2 \leq x_i \leq 10 \text{ for } i = 1, 2, 3\end{gathered} \tag{5.3}$$

Using the simplical CHIM 120-point ansatz, we obtained a smooth, connected Pareto front (figure 5.3) covering most of the first octant using four iterations of our algorithm and a few thousand function evaluations.

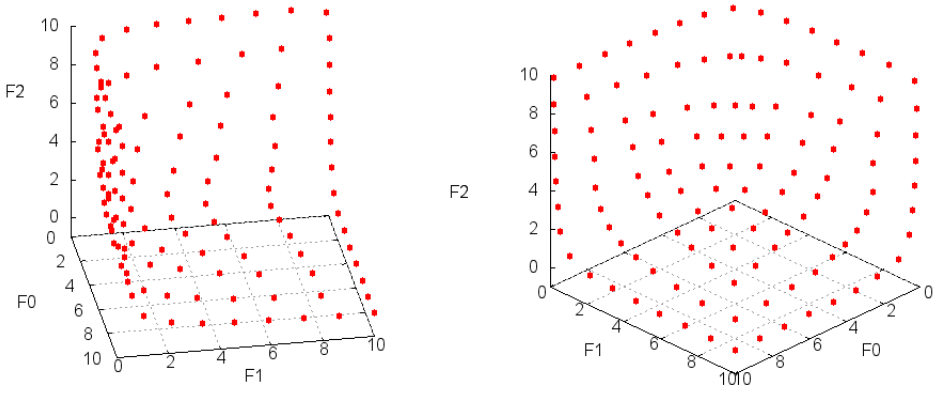

Figure 5.3: Pareto front for problem (5.3) consisting of 120 points obtained with 6,264 function evaluations.

We computed hypervolume and evenness values, as well as state the number of function evaluations required per Pareto point, and compare them to reported values from the literature in table 5.1.

Table 5.1: Problem (5.3) Results

Method	**E**	**FC/Point**
WS	4.331	120.2
NBIm	0.2958	34.3
NCm	0.2958	34.4
Algorithm 3.1	0.01930	52.2

A comparison of solution quality (evenness [**E**]) and computational effort (function evaluations per point [FC/Point]) for problem (5.3). Reference values for WS, NBIm, and NCm algorithms were obtained from [23]. It is not clear whether the number of function evaluations reported includes those induced by finite difference schemes, or if analytical derivatives were used in the scalar minimizations. Our results were obtained with a first-order finite difference implementation.

5.2.2 Problem 2

The other problem is an extension of (5.3) to four objectives.

$$\begin{gathered}\min_{\mathbf{x}} f_i(\mathbf{x}) = x_i \text{ for } i = 1,2,3,4 \\ \text{subject to} \\ -x_1 + x_2^{-1} + x_3^{-1} + x_4^{-1} \leq 0 \\ x_1^{-1} - x_2 + x_3^{-1} + x_4^{-1} \leq 0 \\ x_1^{-1} + x_2^{-1} - x_3 + x_4^{-1} \leq 0 \\ x_1^{-1} + x_2^{-1} + x_3^{-1} - x_4 \leq 0 \\ 0.2 \leq x_i \leq 10 \text{ for } i = 1,2,3,4\end{gathered} \tag{5.4}$$

Likewise, with a simplical CHIM 220-point ansatz, we obtained a smooth, connected Pareto front (figure 5.4) using two iterations of our algorithm.

As before, we computed hypervolume and evenness values, and tracked the number of function evaluations required per Pareto point. Table 5.1 compares the results to reported values from the literature.

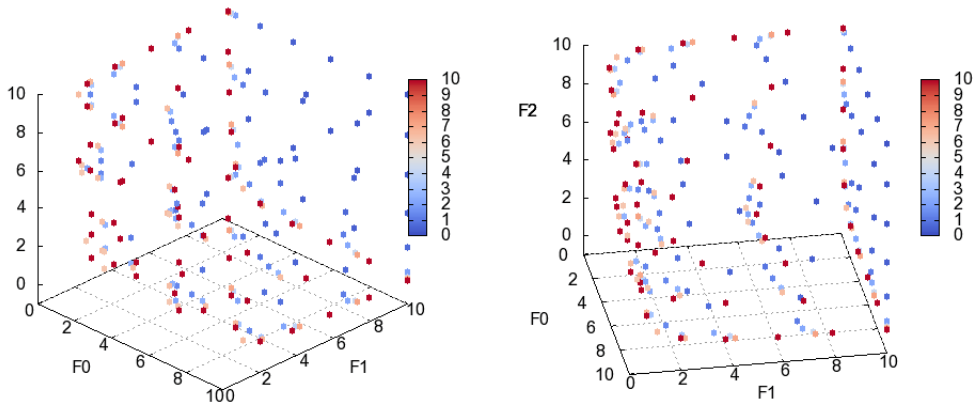

Figure 5.4: Pareto front for problem (5.4) consisting of 220 points obtained with 10,670 function evaluations. The value of the fourth objective for each point is shown using the red-blue color color-map.

Table 5.2: Problem (5.4) Results

Method	**E**	**FC/Point**
WS	4.836	231.9
NBIm	0.3262	49.3
NCm	0.3072	55.3
Algorithm 3.1	0.02091	48.5

A comparison of solution quality (evenness [**E**]) and computational effort (function evaluations per point [FC/Point]) for problem (5.4). Reference values for WS, NBIm, and NCm algorithms were obtained from [23].

5.2.3 Problem 3

On the other hand, the EA community tends to focus on different kinds of problems. EAs, being driven by random generation and recombination of solutions, are typically useful in situations where derivative information is unavailable or unhelpful. As a result, test problem suites intended for EAs employ highly

non-convex, rapidly oscillating, and extremely discontinuous objectives that are often inappropriate for constraint-based methods, like the one presented here. Therefore, we limit our discussion to a representative example as the primary motivation is to demonstrate the precise equispacing of produced solutions and economy of function evaluations.

$$\begin{gathered}
\min_{\mathbf{x}} f_i(\mathbf{x}) \text{ for } i = 1, 2, 3 \\
\text{with} \\
f_1(\mathbf{x}) = (1 + g(x)) \cos(x_1 \pi/2) \cos(x_2 \pi/2) \\
f_2(\mathbf{x}) = (1 + g(x)) \cos(x_1 \pi/2) \sin(x_2 \pi/2) \\
f_3(\mathbf{x}) = (1 + g(x)) \sin(x_1 \pi/2) \\
g(\mathbf{x}) = \sum_{i=2}^{d} (x_i - 0.5)^2 \\
\text{with} \\
0 \leq x_i \leq 1 \text{ for } i = 1, 2
\end{gathered} \tag{5.5}$$

We selected a workable problem, defined in (5.5), from a standard EA test suite [8] and compare the performance to some other heuristics noted in the literature.

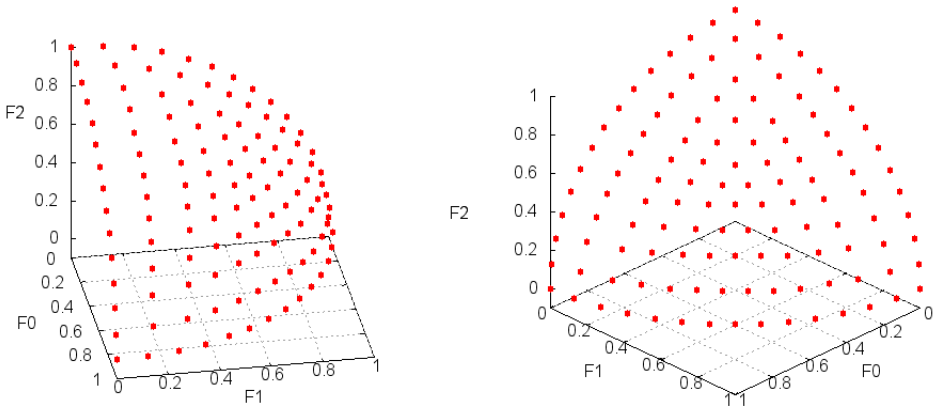

Figure 5.5: Pareto front for problem (5.5) consisting of 120 points obtained with 15,765 function evaluations. The finite differences scheme coupled with the fact that the problem has 10 independent variables gives rise to the very high number of function evaluations required. For examples of typical fronts generated by NSGA-II and SPEA2, see [8].

The resulting Pareto front, as seen if figure 5.5, is computed using significantly fewer function evalutations, obtains a much better hypervolume than any of the algorithms compared in either [25] or [20], and displays more regular sampling than [8].

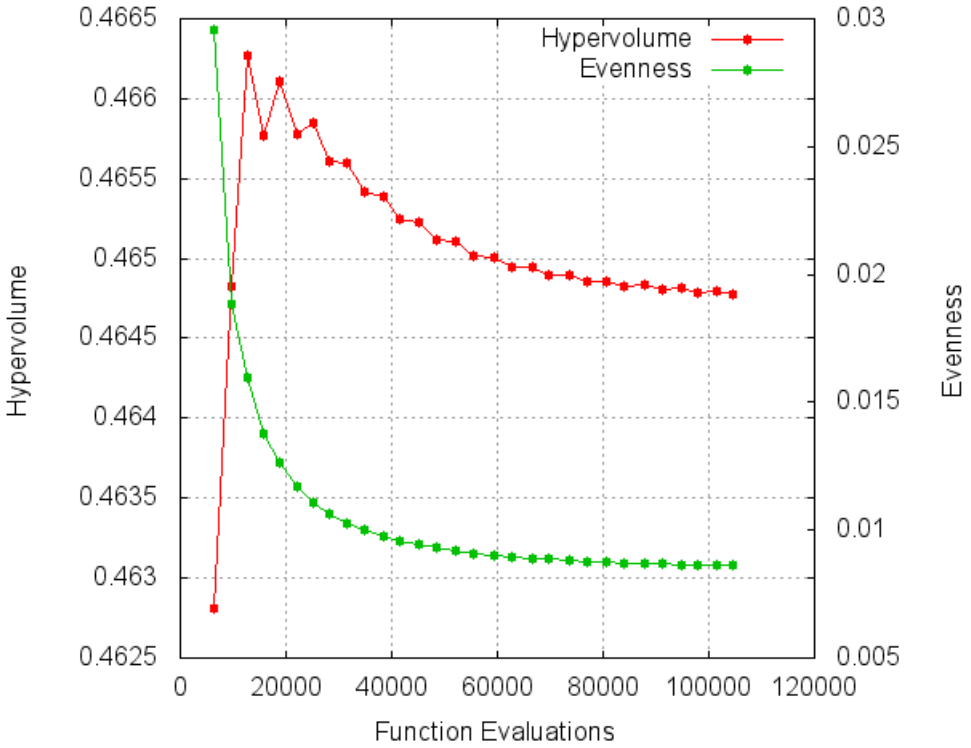

Figure 5.6: Solution quality metrics for Pareto front for problem (5.5). Both the solution hypervolume and evenness improve asymptotically as the number of function evaluations increase. In this example, one iteration of algorithm 3.1 corresponds to about 3,000 function evaluations. The front hypervolume is computed with respect to the origin.

In figure 5.6, we demonstrate the aymptotic improvement in both hypervolume (computed with respect to the origin) and evenness. This implies that, for certain simple front topologies, algorithm 3.1 can be terminated after only a few iterations. Moreover, figure 5.6 shows that further iterations of the algorithm tend to improve the spacing of points more than the obtained hypervolume. This is a result of directly running a set of scalar optimizations (allowing samples to converge to the true optimal front more quickly) rather than ranking and recombining randomly generated solutions.

Chapter 6

Application

The design and operation of particle accelerators requires balancing a multitude of control inputs to optimize any number of desired output parameters. For instance, when discussing optimial beam dynamics, we aim to minimize the beam dispersion in both position space, RMS_x, and momentum space, RMS_s. Given the control parameters allowed, σ_x and V_acc, this unfortunately involves a trade-off between objectives as they can not be simultaneously minimized. Therefore, we aim to sample the Pareto front spanning these conflicting design criteria. We will leverage the Object Oriented Parallel Accelerator Library, OPAL [1], to simulate the charged particle optics.

The result is a simple bi-objective optimization problem with two independent variables. The real complexity of this problem lies in the fact that, while considerable effort has been spent on reducing the cost of simulating the beam dynamics through parallelization [12] and other adaptive techniques, the forward solver mapping the design variables to the objective space remains computationally expensive. Therefore, we will seek to reduce the number of required forward solves as much as possible.

First, as in [2], we run a set of simulations (in our case, approximately 330) corresponding to a grid in the design space. We then construct a Radial Basis Function Network (RBFN) surrogate [30] using these simulations.

After building an RBFN interpolation for each output variable, we applied algorithm 3.1 to the resulting bi-objective problem. Figure 6.2 illustrates the obtained discrete Pareto front representation.

To demonstrate the strong scalability of our approach, we repeated the same calculation while varying the processor count. We performed the following experiments on the Brutus cluster at ETH Zürich. The cluster consists of 1008 nodes totalling 19,760 processing cores connected by an InfiniBand QDR network and attains a throughput of over 200 teraflops. The main program was compiled with GCC 4.6.2 and executed with the use of the MPICH runtime. The current policy of the Brutus standard queue system allows only 128 cores to be used simultaneously per job. Since the opt-pilot framework maps each process type (Optimizers, Pilots, and Workers) to a different core, each optimization

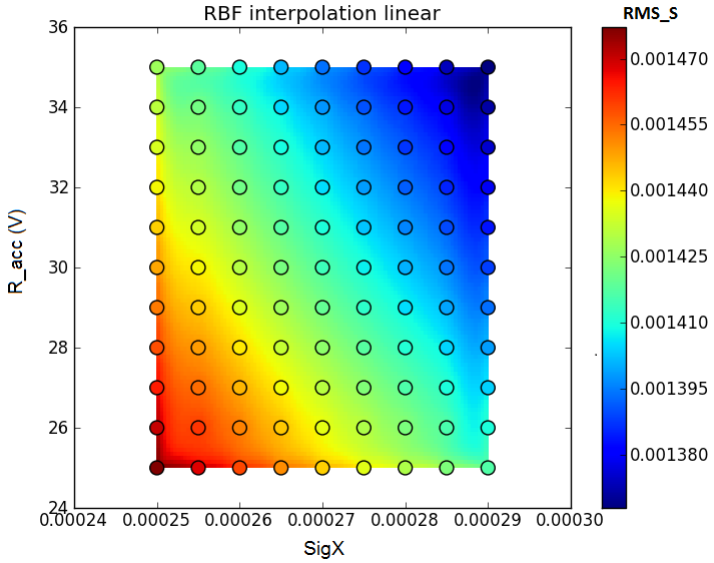

Figure 6.1: RBFN surrogate function trained with data (shown as dots) obtained by running OPAL-based forward solves on a regular set of design variables. Note the fidelity and smoothness of the interpolation.

instance really requires three processors. Therefore, the maximum number of simultaneous optimization instances we are able to run in this configuration is 42. The asynchronous message exchange between processors introduces a small bit of non-determinism and therefore scatter in our data, which was mitigated performing five runs and averaging the results, reported in table 6.1.

Figure 6.3 represents the speedup results in graphical form and compares them to perfect strong scalability. We were limited in our scalability tests by the alloted core limits on the Brutus cluster. However, since we only computed 120 sample points, the roll-off at the end of the range may be attributable to poor load balancing. In our case, some processors will only have two points to optimize, while others have three, resulting in a 50% overload.

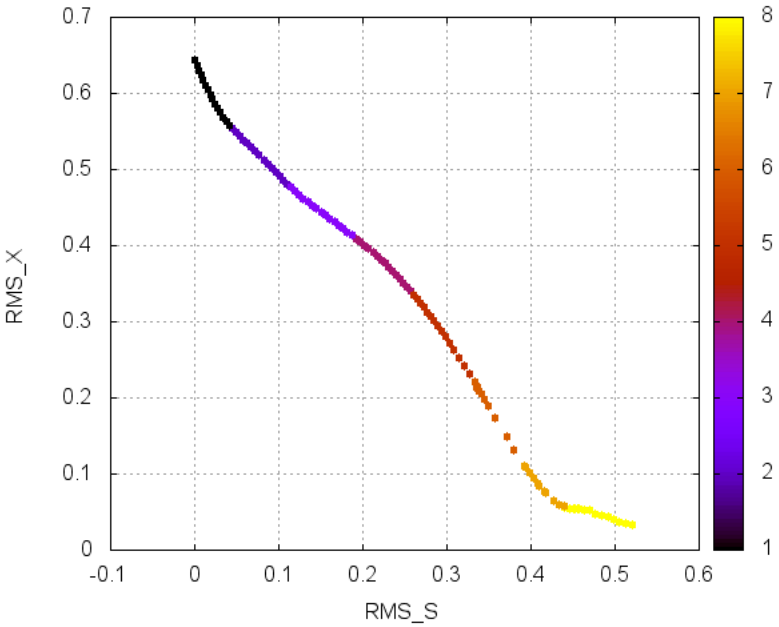

Figure 6.2: Pareto front consisting of 120 points and showing the trade-off between minimizing RMS_s and RMS_x. The point colors indicate mesh-subset membership or the processor with which each sample was obtained. This particular result was computed across 8 different minimization processes running on the Brutus cluster at ETHZ.

Table 6.1: Strong Scalability

Processor Count	**Time (s)**	**Speedup**
1	1529.05	1
2	916.831	1.995
4	527.761	3.718
8	318.502	7.223
16	145.563	15.13
32	84.524	29.36
42	70.9816	39.94

Wall-clock time-to-solution for method 3.1 on a bi-objective surrogate-based problem averaged over 5 runs on the ETHZ Brutus cluster.

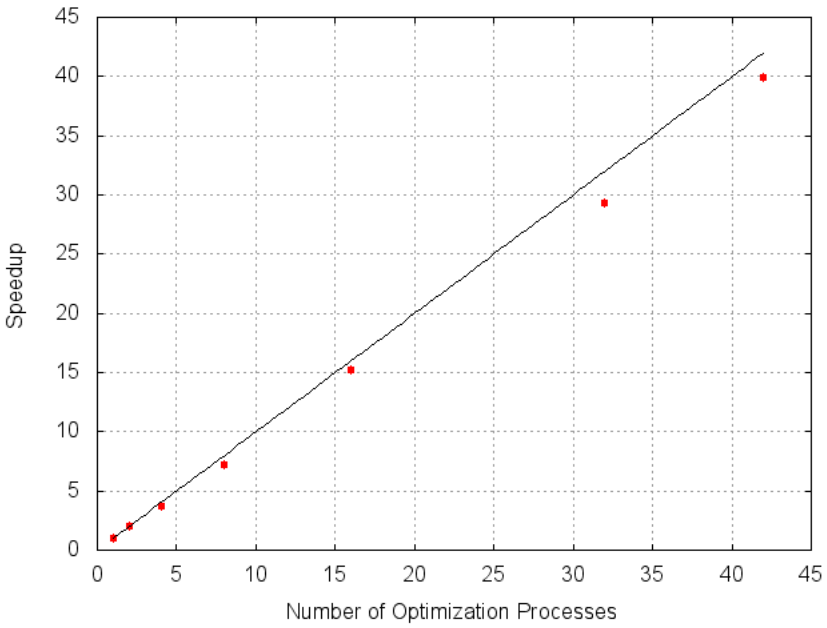

Figure 6.3: The strong scalability of our Pareto front sampling approach (red dots) as compared with a perfect linear speed up (solid line). The extent of our scaling test was limited by the standard core allowances on the Brutus cluster.

Chapter 7

Conclusions & Future Work

Here, an improved algorithm for multi-objective optimization that produces uniformly sampled Pareto fronts by construction is presented. While the algorithm is general, it is most suitable for application to simulation-based engineering optimization problems where economy of function evaluations and smoothness of result are critical. The algorithm discussed acheives an order of magnitude improvement over other geometrically motivated methods, like Normal Boundary Intersection and Normal Constraint, with respect to solution evenness for similar computational expense. This benefit of the proposed method remains, and even improves, after scaling the number of dimensions (and therefore the difficulty of the problem). Furthermore, the resulting uniformity of solutions extends even to more difficult problems, such as those appearing in common EA test cases.

While the resulting discrete representation of the Pareto front, and computational expense of achieving it, are both improved, the most important aspect of the proposed method is its amenability to parallelization. This was demonstrated in an application to a bi-objective problem in the field of particle accelerator design. We demonstrate excellent strong scalability out to more than forty processing elements.

One unfortunate consequence of using a standard SQP package for our scalarized sub-problems, is that, in the algorithm's process of minimizing the scalarized objective, the constraints are not guaranteed to be satisfied. Therefore, not only does this possibly force us to compute the objective functions for points that may lie outside the feasible region, but also the equispacing constraint must always recognize the previous iterations feasible values of the neighbor nodes. This slows down the diffusion of mesh points and can create localized sampling imbalances, especially in situations where mesh neighbors reside on different processors. This can be seen in figure 6.2 occurring between processors six and seven. This situation could be improved by instead using a Feasible Sequential Quadratic Programming (FSQP) routine [18] to optimize the sub-problems. While these solvers are more computationally expensive, they guarantee the feasibilty of each iteration of the optimization process, allowing truly asynchronous

neighbor updates and, thus, smoother sampling.

Another interesting improvement involves using a dynamic RBFN interpolator to reduce the number of actual function evaluations required. Often, retraining a simple interpolator is much less expensive than even a single function evaluation. As seen in figure 5.6, the bulk of the function evaluations occur when refining the sample points to enforce the equispacing criterion and after the samples have already converged to a decent representation of the optimal front. This means that a large number of function evaluations occur in a relatively small subset of the design space. Rather than using costly forward solvers for these mappings, one could maintain a dynamic RBFN that is constantly updated with the simulation results. Once the interpolation fidelity reaches a predefined threshold and the samples converge to the optimal front, the optimizer would no longer need to send sets of design variables to the workers but only consult the local RBFN. According to figure 5.6, we may be able to eliminate up to 50% of the calls to the expensive forward solver.

Furthermore, the questions raised by mapping the opt-pilot processes to actual hardware resources also deserve closer inspection. In the presented examples, only one worker instance was permitted per optimizer, and certainly better performance would result from exploiting more horizontal parallelism by employing more worker instances. Also, while mapping neighboring optimization processes to adjacent processors is straightforward for low-dimensional problems on two and three dimensional torus networks, determining adjacency mappings will be increasingly difficult once the dimensionality of the problem exceeds that of the network. This could perhaps be addressed by space filling manifolds in higher dimensions.

Finally, since a range of different multi-objective optimization paradigms exists, one should consider hybrid schemes. Combining attributes of EAs and geometrically motivated algorithms may prove beneficial for problems that are either too difficult for the likes of NBI and NC or too large for EAs. As seen earlier, the efficacy of the presented algorithm is heavily influenced by the nature of the ansatz. One could, instead of our proposed generalized ansatz implementation, seed the algorithm with the output of a simple EA so as to create a better starting point for homotopic refinement.

Bibliography

[1] Adelmann, Binder, and Kraus. The Object Oriented Parallel Accelerator Library (OPAL). *Proceedings of the Particle Accelerator Conference 2009*, (2):4461–4463, 2009.

[2] Alonso, LeGresley, and Pereyra. Aircraft design optimization. *Mathematics and Computers in Simulation*, 79(6):1948–1958, February 2009.

[3] Athan and Panos. A note on weighted criteria methods for compromise solutions in multi-objective optimization. *Engineering Optimization*, 1996.

[4] Bleuler, Laumanns, Thiele, and Zitzler. PISA—a platform and programming language independent interface for search algorithms. *Evolutionary multi-criterion optimization*, pages 494–508, 2003.

[5] Conn, Gould, and Toint. A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2):545–572, 1991.

[6] Das and Dennis. Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems. *SIAM Journal on Optimization*, 8(3):631–657, 1998.

[7] Deb and Pratap. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[8] Deb, Thiele, Laumanns, and Zitzler. *Scalable test problems for evolutionary multiobjective optimization.* Springer, 2005.

[9] Deb, Zope, and Jain. Distributed computing of Pareto-optimal solutions using multi-objective evolutionary algorithms. *Proceedings of the Second International Conference on Evolutionary Multi-Criterion Optimization*, 2003.

[10] Durillo, Nebro, Luna, and Alba. A study of master-slave approaches to parallelize NSGA-II. *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, April 2008.

[11] Dutta and Kaya. A new scalarization and numerical method for constructing the weak Pareto front of multi-objective optimization problems. *Optimization*, 2011.

[12] Ineichen, Adelmann, Bekas, Curioni, and Arbenz. A fast and scalable low dimensional solver for charged particle dynamics in large particle accelerators. *Computer Science - Research and Development*, May 2012.

[13] Ineichen, Adelmann, Bekas, Curioni, and Arbenz. A Massively Parallel General Purpose Multi-Objective Optimization Framework Applied to Beam Dynamic Studies. *Proceedings of the International Conference of Atomic Physics 2012*, (1):62–66, 2012.

[14] Ineichen, Adelmann, Bekas, Curioni, and Arbenz. A Parallel General Purpose Multi-Objective Optimization Framework, Applied to Beam Dynamic Studies. *Proceedings of the International Conference of Atomic Physics 2012*, pages 1–18, 2012.

[15] Ishibuchi, Tsukamoto, Hitotsuyanagi, and Nojima. Effectiveness of scalability improvement attempts on the performance of NSGA-II for many-objective problems. *Proceedings of the 10th annual conference on Genetic and evolutionary computation - GECCO '08*, page 649, 2008.

[16] Jensen. Reducing the Run-Time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms. *IEEE Transactions on Evolutionary Computation*, 7(5):503–515, October 2003.

[17] Kraft. Algorithm 733: TOMP–Fortran modules for optimal control calculations. *ACM Transactions on Mathematical Software (TOMS)*, 20(3):262–281, 1994.

[18] Lawrence and Tits. A Computationally Efficient Feasible Sequential Quadratic Programming Algorithm. *SIAM Journal on Optimization*, 11(4):1092–1118, January 2001.

[19] Leyffer. A Complementarity Constraint Formulation of Convex Multiobjective Optimization Problems. *INFORMS Journal on Computing*, 21(2):257–267, September 2008.

[20] Li, Kwong, Cao, Li, Zheng, and Shen. Achieving balance between proximity and diversity in multi-objective evolutionary algorithm. *Information Sciences*, 182(1):220–242, January 2012.

[21] Marler and Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, April 2004.

[22] Messac and Mattson. Normal constraint method with guarantee of even representation of complete Pareto frontier. *AIAA journal*, 42(10):2101–2111, 2004.

[23] Motta, Afonso, and Lyra. A modified NBI and NC method for the solution of N-multiobjective optimization problems. *Structural and Multidisciplinary Optimization*, 46(2):239–259, January 2012.

[24] Mueller-Gritschneder. A successive approach to compute the bounded Pareto front of practical multiobjective optimization problems. *SIAM Journal on Optimization*, 20(2):915–934, 2009.

[25] Nebro, Durillo, and Coello. A study of convergence speed in multi-objective metaheuristics. *Proceedings of the 10th international conference on Parallel Problem Solving*, 2008.

[26] Nebro, Durillo, Luna, Dorronsoro, and Alba. MOCell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems*, 24(7):726–746, July 2009.

[27] Pereyra. Fast computation of equispaced Pareto manifolds and Pareto fronts for multiobjective optimization problems. *Mathematics and Computers in Simulation*, 79(6):1935–1947, February 2009.

[28] Pereyra, Saunders, and Castillo. Equispaced Pareto front construction for constrained bi-objective optimization. *Mathematical and Computer Modelling*, pages 1–10, 2011.

[29] Powell. Direct search algorithms for optimization calculations. *Acta Numerica*, 7, 1998.

[30] Queipo, Haftka, Shyy, Goel, Vaidyanathan, and Tucker. Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, 41(1):1–28, January 2005.

[31] Smith and Everson. Dominance-based multiobjective simulated annealing. *IEEE Transactions on Evolutionary Computation*, pages 1–20, 2008.

[32] While, Bradstreet, and Barone. A fast way of calculating exact hypervolumes. *IEEE Transactions on Evolutionary Computation*, 16(1):86– 95, 2012.

[33] Zitzler and Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.