

Iteration-wise Intermediate Points Saved in an Archive as Outcome of an Optimization Algorithm Considered Unreliable

COIN Report 2026010

Kalyanmoy Deb^a, Ahmer Khan^b

^a*Department of Electrical and Computer Engineering Michigan State
University East Lansing MI 48824 USA*

^b*Department of Computer Science and Engineering Michigan State
University East Lansing MI 48824 USA*

Abstract

A numerical optimization algorithm starts with a single or multiple initial points and in every iteration moves to a new point or a population of points depending on the transition operations involved. The goal of an optimization algorithm is to arrive at the final ‘optimized’ solution(s) close to the true optimal solution(s). The iteration-wise intermediate points that an algorithm create are stepping-stones to arrive at the final optimized solution(s). In this paper, we strongly advocate of not using the iteration-wise intermediate solutions save in a passive archive as the final outcome of the optimization task. In a task of finding multiple optimal or targeted solutions for a problem, such as, solving multi-modal optimization problems in which the goal is to find multiple local and global optimal solutions, or solving multi-objective optimization problems, in which the goal is to find multiple Pareto-optimal

Email addresses: kdeb@msu.edu (**Kalyanmoy Deb**), khanahm2@msu.edu (**Ahmer Khan**)

solutions, one should not consider the iteration-wise intermediate points, save in an archive, and declare one or more of the archive members as the final outcome of the ensuing optimization algorithm. This is because the intermediate points were not the original intended goal and in the event of them being close to any true optimal solution is a sheer happenstance. In this paper, we explain this important philosophical aspect of various optimization tasks with examples and clearly argue why optimization researchers should avoid considering non-interactive archive points as the final outcome. We also highlight some acceptable use of iteration-wise intermediate points in gathering knowledge and constructing more sophisticated optimization algorithms.

Keywords:

Genetic algorithms, intermediate points, optimization algorithm, optimized solution

1. Introduction

An optimization algorithm usually works in iterations starting with an initial solution (a vector of decision variables or a *point* in a multi-dimensional space) or a set of solutions/variable vectors/points, collectively called as a population. The goal of an optimization Algorithm is to find one or more optimal solutions ($\mathbf{x}^* \in \mathbb{R}^n$) of the underlying problem, given below, formulated with one or more objective functions ($\mathbf{f}(\mathbf{x}) \in \mathbb{R}^M$) to be minimized and a number of constraint functions (inequality constraints $\mathbf{g}(\mathbf{x}) \in \mathbb{R}^J$ and equality constraints $\mathbf{h}(\mathbf{x}) \in \mathbb{R}^K$) and variable bounds ($x_i \in [x_i^{(L)}, x_i^{(U)}]$) for

each variable) to be satisfied (1; 2; 3):

$$\begin{aligned}
 & \text{Minimize } f_m(\mathbf{x}), & m = 1, 2, \dots, M, \\
 & \text{subject to } g_j(\mathbf{x}) \leq 0, & j = 1, 2, \dots, J, \\
 & h_k(\mathbf{x}) = 0, & k = 1, 2, \dots, K, \\
 & x_i^{(L)} \leq x_i \leq x_i^{(U)}, & i = 1, 2, \dots, n.
 \end{aligned} \tag{1}$$

Primarily, optimization algorithms can be broadly categorized based on the number of points used in each iteration. Point-based algorithms use a single point in each iteration and proceed towards a single optimal solution (1; 4) with iterations. Population-based algorithms work with a set of points (a population) in each iteration and proceed towards a single optimal solution for single-objective optimization tasks and proceed towards multiple optimal solutions (in multi-modal or multi-objective problems) parallelly by maintaining a diverse population (5; 6; 3; 7), thereby finding multiple optimal solutions in a single simulation run.

It is clear that at the end of every iteration, an optimization algorithm finds a new intermediate point or a population of intermediate points, only as a stepping-stone to move to better points in the subsequent iterations. A history of iteration-wise intermediate points saved in an archive indicates how an optimization algorithm has progressed from the starting point(s) to near optimal solution(s) with iterations. From a systems approach perspective, an optimization task can be viewed as one that takes the problem formulation and initial point(s) as input, and churns out through a series of iterative computations one or more optimal solutions as output (or outcome). What happens inside the system are specific to the structure of the chosen optimization algorithm and should not be considered as an outcome

of the system. This is so due to a number of reasons. First, a change in the initial point should still meet the goal of the optimization task of finding one or multiple optimal solutions. However, iteration-wise intermediate points largely depend on the chosen initial point(s) and hence the resulting optimization Algorithm that considers intermediate points in deciding the outcome of the algorithm may not constitute a reliable approach. Second, the change of an optimization algorithm should still meet the goal of the optimization task. Since every optimization algorithm has a different working principle, the iteration-wise intermediate points are likely to be different, despite their end outcome being identical or similar.

Despite these basic understanding of the goal of an optimization task and what is expected from an optimization algorithm, some optimization studies follow a strategy in which all iteration-wise intermediate points are saved in an external archive for a post-optimal analysis without any involvement of the archive in creating new solutions in the future iterations. A subset selection strategy is then applied to pick some solutions from the final external archive (8; 9), irrespective of which iteration these solutions were saved into the archive. We argue that such a strategy is unreliable, unrepeatable, unintelligent, and relies on happenstance, and lacks an algorithmic sanctity.

Having said that, we realize that such a strategy may be a good practice for a one-time optimization task in practice in which solution evaluation is overly expensive and there is no chance of arriving at the true optimal solution with a limited number of solution evaluations. In such low-budget optimization applications, every high-fidelity evaluated solution, intermediate or final, must be considered before declaring the final outcome of the opti-

mization algorithm, fully knowing that this solution may not be even close to true optimum. But a typical optimization task trying to establish the efficacy of its carefully designed and orchestrated operators may need to be use a reasonably large computational budget to discover the true optimal or near-optimal solutions of the problem and compares its optimized solutions with that of other competing algorithms to establish its superiority, reliability, repeatability, and intelligent operator coordination. The low-budget optimization algorithms mentioned above should not be compared with these typical relatively large-budget optimization algorithms, which are designed to achieve a completely different goal.

Putting the unreliable and unrepeatable aspect of the resulting optimization algorithm aside, one may argue that there is no harm in considering intermediate points along with the final solutions for deciding the outcome of an optimization algorithm. If no good intermediate points exist, none of them will be selected as the final outcome anyway. As we demonstrate in this paper, for some optimization algorithms, such a sloppy procedure may help select certain unintended solutions (which may be close to the desired solutions) by chance, but such an approach is certainly not reliable and repeatable and in some occasions may violate the theory or fundamental understanding of the working of the algorithm. In the event of a multi-solution goal of the optimization task, a search for one specific optimal solution may accidentally discover a completely different and unintended optimal solution. Accepting such a "surprised" optimal solution does not constitute a respectful and theoretically sound and intelligent algorithm. Because the intermediate points are merely stepping stones towards arriving at the final optimal solutions,

optimization algorithms are never designed to have any specific intermediate point as a target. Moreover, the intermediate points intricately depend on the initial points with which the algorithm was started. A counter argument may be posed instead; if an intermediate point has to be a part of the final outcome, why did not the optimization algorithm aim at finding it at the first place?

In the remainder of this paper, we discuss a number of standard optimization tasks, by clearly highlighting the goals of each of them, in Section 2. Section 3, we discuss standard reformulations of the optimization problem, for one type of optimization problems, so that the desired goals can be achieved as an outcome of the resulting reformulated optimization tasks. Then, in Section 4, we discuss a few reasons why iteration-wise intermediate points cannot be considered for declaring the final outcome of an optimization algorithm in a reliable manner. Section 5 presents various different optimization tasks and makes the argument further with simple illustrative test and engineering examples. Based on the presented results, Section 6 suggests unacceptable optimization methods and a number of acceptable optimization strategies, some of which are already practiced in the optimization literature. The discussion should guide optimization researchers to propose reliable, repeatable, and respectful optimization algorithms. Finally, conclusions of this study are drawn in Section 7.

2. Types of Optimization Tasks and Their Outcomes

Optimization tasks come in different forms. Some of them are mentioned below.

- **Unconstrained versus constrained optimization:** If no constraints and variable bounds are present in an optimization problem, the task becomes an *unconstrained* optimization. Otherwise, it is known a *constrained* optimization (1; 2). However, if only variable bounds are present without any equality or inequality constraints, the task is known as *optimization with box constraints*. The goal of constrained optimization is to find a feasible solution, which satisfies all constraints and the specified variable bounds, and minimizes the objective function. Thus, the outcome of an unconstrained optimization and constrained optimization can be quite different.
- **Uni-modal versus Multi-modal optimization:** If the goal of an optimization task is to find a single optimal solution, despite the presence of multiple local and global optimal solutions in the search space, the task is referred to as a *uni-modal* optimization task. On the other hand, if the goal is to find multiple local and/or global optimal solutions, the task is referred to as *multi-modal* optimization (10; 11). Thus, a multi-modal optimization refers to finding not one, but multiple, local or global optimal solutions.
- **Single versus multiple objective optimization:** If the optimization problem involves a single objective function for minimization or maximization, the task is referred to as *single-objective* optimization, else it is generally called a *multi-objective* optimization (7; 3; 12), despite a recent further classification resulting in many-objective optimization for problems having more than three objectives (13; 14; 15).

In a single-objective optimization, usually a single optimal solution is targeted, and in a multi-objective (or many-objective) optimization, multiple Pareto-optimal solutions (with trade-offs in objectives) are the target.

- **Structured versus generic nonlinear optimization:** If an optimization problem has a simplistic structure (1), such as, all objectives and constraints being linear functions of the variables, the task is called a *linear programming* (LP) task. If the objective function is quadratic and constraints are linear, it is called a *quadratic programming* (QP) task. If the objective function is convex and the feasible space defined by the constraints is convex, the task is called a *convex programming* task. While the scope of structured problems are constantly being enhanced, such as, for difference-in-convex functions (16), the problems which do not fall under a structured programming type is referred to as a *generic* nonlinear programming or an *unstructured* optimization task. The outcome of a structured programming algorithm is usually a single global optimal solution, however, a generic nonlinear optimization problem may have multiple local and global optimal solutions and non-convex and disjointed feasible regions. Thus, clearly generic non-linear programming problems are difficult to solve and the No-Free-Lunch theorem (17; 18) states that there cannot exist a single optimization algorithm most efficient in solving an arbitrary generic nonlinear optimization problem.

- **Stationary versus dynamic optimization:** If an optimization problem has non-changing objectives and constraint functions, and variable bounds, the task is referred to as *stationary* optimization. Otherwise, any change in any of its components, whether it affects the optimal solution or not, is referred to as *dynamic* optimization (19; 20; 21). In a stationary optimization, the outcome is usually one or more solutions in the search space, whereas in dynamic optimization, the optimal solution set usually changes with time (or iteration). Hence, the target for a dynamic optimization algorithm is to reach the true optimal solutions as closely as possible, till the components of the problem change, making these problems more difficult to solve.
- **Single-level versus multi-level optimization:** If an optimization task is defined in a single level without resorting to solve any other nested optimization task, it is referred to as a *single-level* optimization. Otherwise, it is referred to as a *multi-level* optimization. Bi-level optimization involves two optimization problems (22; 23; 24), in which for every upper-level solution, a distinct lower-level optimization task needs to be solved in a nested manner. These problems commonly appear in practical problems, such as, truss structure design, engineering systems design, etc. A multi-level optimization must coordinate more than one nested optimizations well to arrive at a combination of optimal solutions for a hierarchy of objectives on multiple levels.

Optimization tasks can be of a mixed type, such as, dealing with a single-level, constrained, multi-modal, multi-objective, nonlinear, and stationary optimization problem. Although all the qualifiers mentioned above are usu-

ally stated to describe an optimization task, some of them are usually ignored to indicate some default types. For example, the above task may be referred commonly as a multi-modal, multi-objective optimization task (25), keeping the default notion that it involves a single-level, constrained, nonlinear and stationary optimization task, resulting in the search of finding multiple Pareto-optimal fronts (PFs) of solutions.

In addition, based on the types of the variables, the optimization task can be real-valued, discrete, combinatorial, or mixed type. Each type of an optimization task, mentioned above, needs a specialized optimization algorithm to suit the search process. In addition, there are two ways of classifying optimization algorithms, which are worth mentioning:

- **Deterministic versus stochastic optimization algorithms:** This is referred mainly for the chosen optimization algorithm. If an algorithm produces an identical sequence of points starting from an identical initial point, it is referred to as *deterministic* optimization algorithm. Otherwise, it is known as a *stochastic* optimization algorithm..
- **Point versus population optimization algorithms:** In a broad sense, if every iteration of an optimization algorithm works with a single point and is updated to a new single point, it is referred to as *point-based* optimization algorithm. If multiple points are employed in every iteration, it is referred to as a *population-based* optimization algorithm. Evolutionary optimization algorithms (5; 6) are population-based methods.

Note that each of the optimization tasks mentioned earlier can be solved by

using either a deterministic or stochastic, and a point or population-based algorithm.

3. Concept of a Reliable and Repeatable Optimization Algorithm

Different optimization tasks aim for different outcomes, as outlined above, but in order for an optimization algorithm to be reliable and repeatable, it should produce a similarly good solution close to the true optimal solution(s) every time it is executed.

We discuss this aspect through the task of uni-modal optimization. The goal in a uni-modal optimization task is to find a single optimal solution (local or global) for an optimization problem and thus, the outcome is one local or one global optimal solution. Note that for differentiable problems, in which the objective and all constraint functions are differentiable, a local-optimal solution must satisfy a set of Karush-Kuhn-Tucker (KKT) optimality conditions (4; 2). However, a strategy to discover optimal solutions using the optimality conditions is not commonly used due to the involvement of derivatives and requirement of satisfying a number of equality and inequality conditions, besides meeting a constraint qualification condition (1). Moreover, KKT conditions are only the necessary conditions for a solution to be a candidate and second-order optimality conditions must also be satisfied for a solution to be minimum.

Starting from an arbitrary initial point, a point-based optimization algorithm, generates a new and hopefully an improved point in each iteration and progress toward a local/global optimal solution with iterations. Despite creating a new and improved point in each iteration, the outcome of a uni-

modal optimization task is the final optimized solution, which could not be improved any further by the algorithm. An optimality check can be performed at the end of each iteration and decide whether to terminate a run or continue, but optimality checks are computationally expensive and usually avoided. Instead, algorithms are run until a fixed number of iterations are elapsed or no substantial improvement in the solution quality has been achieved in the past few iterations.

Most point-based uni-modal optimization algorithms are designed to monotonically non-deteriorate iteration-wise points from start to the end. In this case, a termination at any iteration results in the best-found solution as the outcome of the process. But certain population-based optimization algorithms, specifically stochastic algorithms, may not always improve or non-deteriorate the best intermediate solutions from one iteration to the next. It could be the stochasticity of the optimization process or a purposeful deterioration to overcome a hill before next phase of improvement. Whatever is the case, such non-monotonic algorithms must decide what points from current iteration to carry over to the next iteration judiciously so that the algorithm purposefully progress toward the optimal solutions. Points which are left behind are not considered competitive at the current iteration, but if a later iteration considers these points to be competitive again, they must be created again by the algorithm's operators. This process allows an algorithm to stay its best at any iteration and an arbitrary termination is expected to represent the best solutions found thus far. This aspect keeps the optimization algorithm's operations meaningful at every iteration. As we will discuss in the next section, keeping an archive of all intermediate solutions from start

to end and then analyzing them to pick the best solution from them do not produce a reliable, repeatable and respectful algorithm.

This argument for declaring the outcome of an optimization task from the final iteration of an optimization run can be extended to various types of tasks, mentioned above. For example, in a multi-objective optimization task, the outcome is a finite set of Pareto-optimal solutions. Even in this case, for a reliable algorithm, all potentially non-dominated and less-crowded points from intermediate iterations must be identified and carried forward in an algorithmic manner so as to monotonically and reliably improve the algorithm's performance with iterations and converge to final Pareto-optimal set.

4. Reasons Against Choosing Iteration-wise Intermediate Points

We summarize here the difficulties in selectively choosing iteration-wise intermediate points from an optimization run to declare as desired outcome.

- **Intermediate points are dependent on the initial point:** Every optimization algorithm starts from one or more initial points. The trajectory of intermediate points to arrive at the optimal solution depends on the chosen initial point(s), despite the final outcome being a near-optimal solution. Since the sequence of intermediate points are never the target of an optimization algorithm, every run starting with a different initial point is expected to take a different path (of points) to arrive near the optimal solution. Thus, the iteration-wise intermediate points ideally cannot be relied on for choosing as target solution(s).

- **Intermediate points are dependent on the chosen optimization algorithm:** On the same account, iteration-wise intermediate points from the initial point(s) to the near-optimal solution depends largely on the chosen optimization algorithm. Some algorithm use gradients and find intermediate points along the negative of the first-order gradient directions at the previous point; some algorithms use second-order derivative directions (Newton’s method) to find intermediate points; some use non-gradient approaches, such as conjugate directions (1; 2). Since each optimization algorithm behaves differently, even if they are started from the initial point(s), iteration-wise intermediate points are likely to be different from each other and cannot reliably be called as the outcome of the optimization algorithm.
- **Intermediate point set is different for every run of a stochastic optimization algorithm:** For a stochastic optimization algorithm, the trajectory of intermediate points may not be the same, even if it is started from the same initial point. Hence for stochastic algorithms, intermediate points should never be considered as target solutions.
- **Intermediate points need not be most efficient for desired goals:** A naive technique which stores all iteration-wise intermediate points of the original optimization problem and at the end chooses the specific point(s) that meet new specified optimization goals need not constitute a reliable, repeatable, and respectful procedure. Let us consider a two-step procedure for implementing the above technique. First, a standard optimization algorithm is run and all iteration-wise

intermediate points are stored. Next, specific iteration-wise intermediate points are chosen from the stored points according to the goal of the optimization task. We consider this approach as naive and if it manages to produce an acceptable and desired outcome, is purely a happenstance and it does not constitute a reliable method.

The simplest idea would be to pick the best iteration-wise intermediate point, based on a lexicographic consideration of feasible and smallest (or non-dominated) objective value(s). Such a method is not reliable, due to the reasons outlined in the previous section.

Next, we consider a more involved selection approach, which we argue may not constitute a reliable method as well. Let us consider an optimization task for finding a *robust* solution, which is least sensitive to variable uncertainties but is also locally optimal. Robust solutions, instead of optimal solutions, are often desired in practice. Robustness ($R(\mathbf{x})$) of a solution can be defined as the average penalized objective value of H neighboring solutions ($\mathcal{B}_{\mathbf{x}}$) including \mathbf{x} :

$$R(\mathbf{x}) = \frac{1}{|\mathcal{B}_{\mathbf{x}}|} \sum_{\mathbf{y} \in \mathcal{B}_{\mathbf{x}}} \left(f(\mathbf{y}) + r \left(\sum_{j=1}^J \langle \hat{g}_j(\mathbf{y}) \rangle + \sum_{k=1}^K |\hat{h}_k(\mathbf{y})| \right) \right), \quad (2)$$

where r is a suitable penalty parameter and $\hat{\mathbf{g}}$ and $\hat{\mathbf{h}}$ are normalized constraint vectors. The bracket operator outputs the operand, if the operand is positive, else it outputs zero. The term under the second brackets is often known as the constraint violation (CV) of a solution (26). Other robustness definitions exist in the constrained optimization literature (27; 28; 29), but we use here the above simple definition to make our argument.

Clearly, minimizing $R(\mathbf{x})$, instead of the original sensitive objective function $f(\mathbf{x})$, a robust solution can be found. It is clear that robust solutions can be different from the optimal solution of the original problem. In trying to find the robust solution, a user can follow the naive approach. Run the original optimization task to minimize $f(\mathbf{x})$, but store all iteration-wise intermediate points of a run. Then, instead of declaring the final optimal solution found by the algorithm, the user computes $R(\mathbf{x})$ for every iteration-wise intermediate point as a post-optimal operation. Then, the specific iteration-wise intermediate point having the smallest $R(\mathbf{x})$ value is declared as the robust solution.

We argue here that such a technique is not likely to produce the desired outcome. Never during the optimization algorithm the emphasis was given to find a robust solution! Why would the optimization algorithm produce a robust solution as an iteration-wise intermediate point? If this happens in a particular run, it is a happenstance and the technique will be judged far from being reliable and repeatable. If a robust solution is the target, instead of minimizing the original $f(\mathbf{x})$, we need to minimize $R(\mathbf{x})$ directly, as a solution strategy.

The above naive approach can be explained as an unreliable approach for many other optimization tasks, including discrete optimization, constrained optimization, multi-modal optimization, multi-objective optimization, and others. We discuss more about acceptable and unacceptable optimization methodologies in Section 6.

Despite these issues, we would like to highlight that for *real-world* optimization problem-solving tasks, particularly with time-consuming evaluation

procedures, every feasible solution found by high-fidelity evaluations may be a reasonable candidate as a target solution. Of course, optimization algorithms are expected to produce monotonically non-inferior performance with iterations. So, execution of more iterations are expected to be beneficial, in general, but for some reason if an intermediate point is more acceptable as the final target solution for a real-world scenario, it may be accepted. But certainly, such an approach is not reliable and repeatable as a standard optimization procedure.

5. Results

In this section, we demonstrate the above-mentioned difficulties with the iteration-wise point selection approach in some of the optimization tasks mentioned in Section 3.

5.1. Finding a Robust Solution

To support the argument made in this paper, we first take a single-variable minimization problem:

$$f(x) = 2 - 0.75 \exp\left(-\left(\frac{x + 0.5}{0.7}\right)^2\right) - 0.72 \exp\left(-\left(\frac{x - 0.35}{0.25}\right)^2\right) - \exp\left(-\left(\frac{x - 0.85}{0.05}\right)^2\right), \quad (3)$$

having two local and one global minimum points near $x^* = -0.5$, 0.35 , and 0.70 (global). If our goal is to find the robust solution, rather than an optimal solution, with an uncertainty in the variable in a small neighborhood (say $[-0.05, 0.05]$), then points near the two local minima $x^* = -0.5$ and 0.35 are robust solutions. We create $|\mathcal{B}_{\mathbf{x}}| = 25$ neighboring points within $[-0.05, 0.05]$ of a solution x to compute the robustness metric. Since there is no constraint

for this problem, the second term in Equation 2 is absent. The dashed line plots the robust function $R(x)$ in Figure 1. Figures 1a, 1b and 1c show how the Nelder and Meade optimization algorithm – a population-based deterministic unconstrained optimization algorithm – coded in `fminsearch()` routine of Matlab finds the respective minimal solutions starting from $x = -1$, $x = 0$, and $x = 1$, respectively. Depending on the starting point, the nearest optimal solution (marked with a star) is found. If the intermediate points of

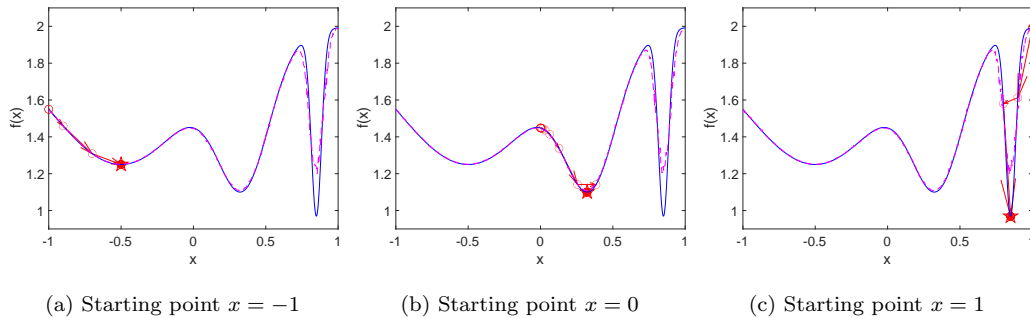


Figure 1: Nelder and Meade optimization algorithm of Matlab’s `fminsearch()` produces different sequence of intermediate points for a different starting point for minimizing $f(x)$ (Equation 3).

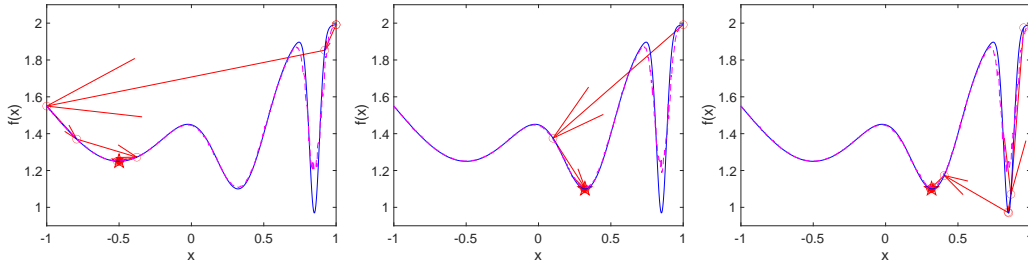
any of these runs are checked with the robustness metric defined in Equation 2, each identifies a different ‘robust’ solution, marked with a solid circle. In each case, the determined ‘robust’ solution is identical to the respective optimal solution. As indicated by the dashed function (obtained using the robustness function), the best robust solution is the point near $x = 0.35$, which is incidentally obtained from the $x = 0$ run (Figure 1b) alone. This clearly indicates that task of reliably finding the robust solution cannot be postponed with a post-optimal task from iteration-wise intermediate points found by a direct optimization of the objective function $f(x)$, as not all runs

find the identical robust solution.

Figure 2a shows the intermediate points for initial point $x = 1$ but optimized with sequential quadratic programming (SQP) algorithm of `fmincon()` routine of Matlab. Comparing Figures 2a and 2b, it can be seen that the two different optimization algorithms starting with the same starting point produce different intermediate points, indicating that the intermediate points produced by an optimization algorithm depend on the algorithm used and cannot be used reliably for choosing for an outcome of an algorithm. However, irrespective of an algorithm's intermediate points, both algorithms seem to have found an identical final solution. Thus, the final solution from an algorithm can be considered as the outcome of the algorithm and the choice of an intermediate point need not produce a reliable and repeatable outcome.

Figure 2c shows the progress of the best-population solution of a real-coded genetic algorithm (RGA) (30) with 12 population members, crossover probability of 0.9, mutation probability of 1.0, SBX index of 2, and polynomial mutation index of 20 (3), in which initial population is set within $x \in [0.95, 1.00]$. The RGA run finds one of the robust solutions, despite traversing through the global minimum, which is not the robust solution for this problem.

The RGA run brings out another interesting aspect. If all intermediate points (best-generation solutions) are stored, despite solving the robust function given in Equation 2, the RGA can be claimed to have found the global optimal solution during an intermediate iteration. But the RGA was not intended to find the global minimum of the original objective function,



(a) SQP path with starting point $x = 1$ and minimizing $f(x)$ directly (b) `fmincon()` finds robust solution directly using $R(x)$ starting with the robust solution directly (c) A genetic algorithm run to finds $x = 1$ directly

Figure 2: Effect of different algorithms and a direct approach for finding the robust solution for problem in Equation 3. Figures show different algorithms find different final solutions starting with the same starting point or population. Figures (b) and (c) discovers the true robust solution at the end, whereas the GA in (c) also finds the global minimum as an intermediate point, but the GA should not be credited for this finding, as it is a pure happenstance.

rather it was intended to find the minimum of $R(x)$ – the robust solution. It did not terminate at this intermediate iteration and is continued to find the robust solution at its final iteration. The post-optimal discovery of the global optimum of the original function, as one of the intermediate points obtained from the robust optimization, should not be used to give credit to the RGA as a global optimizer of the original function. It is a pure happenstance, as starting with a different initial population may have taken the same RGA to follow a different path of arriving at the robust solution. It is the final outcome with which an optimization algorithm should be evaluated, rather than showcasing the accidental finding of an intermediate solution having certain property saved in an archive.

5.2. Finding Discrete Solutions

We consider the following two-variable problem to illustrate the difficulties with the iteration-wise intermediate points of a continuous optimization algorithm to find the discrete optimal solution as a post-optimal task:

$$\begin{aligned}
 & \text{Minimize} && f(\mathbf{x}) = 28x_1 - 1000x_2, \\
 & \text{Subject to} && g_1(\mathbf{x}) \equiv x_2 - 0.01(x_1 + 2)^2 \leq 0, \\
 & && g_2(\mathbf{x}) \equiv 2x_1 + x_2 - 60 \leq 0, \\
 & && g_3(\mathbf{x}) \equiv x_1 - x_2 - 19 \leq 0, \\
 & && x_1 \in [-40 : 5 : 30], \quad x_2 \in [-10 : 2.5 : 20].
 \end{aligned} \tag{4}$$

Variables x_1 and x_2 are allowed in steps of 5 and 2.5 units, respectively, making it a discrete optimization problem. Figure 3a shows the feasible region which lies at the bottom part of the plot surrounded by three constraint boundaries. The optimal solution of the continuous version of the problem is at $x^{*,\text{cont}} = [28, 9]$ with $f^{*,\text{cont}} = -8, 216$. However, since this solution is not an allowable discrete solution (x_1 and x_2 are not a multiple of 5 and 2.5 units, respectively, from their lower bounds), it is not the desired optimal solution. The problem is tricky since the global minimal region does not have a good neighboring discrete solution, whereas the local minimal solution of the continuous version ($\mathbf{x}^* = [-26.9, 6.20]$ having $f = -6, 200$) has a neighboring discrete solution $\mathbf{x}^{*,D} = [-25, 5]$ having $f^* = -5, 700$, making it the optimal solution of the discrete version of the problem.

Figure 3a shows the intermediate and final points obtained when the above problem solved using Matlab's `fmincon()` routine starting from $[5, -8]$. If every intermediate point of the `fmincon()` run is approximated to its nearest discrete variable vector (shown in solid (feasible) and open (infeasible)

blue circles in the figure), none of them comes close to the true discrete optimal solution $\mathbf{x}^{*,D}$. The discrete approximation of intermediate points sometimes produces infeasible solutions, as shown in the figure with open blue circles. However, as shown in Figure 3b, for some initial points (such as, $[-20, -8]$), the discrete approximation of one of the iteration-wise intermediate points produces the discrete optimal solution ($[-25, 5]$). But such a post-optimal approximation approach is not reliable and chance may dictate whether the discrete optimal solution is found or not in a run.

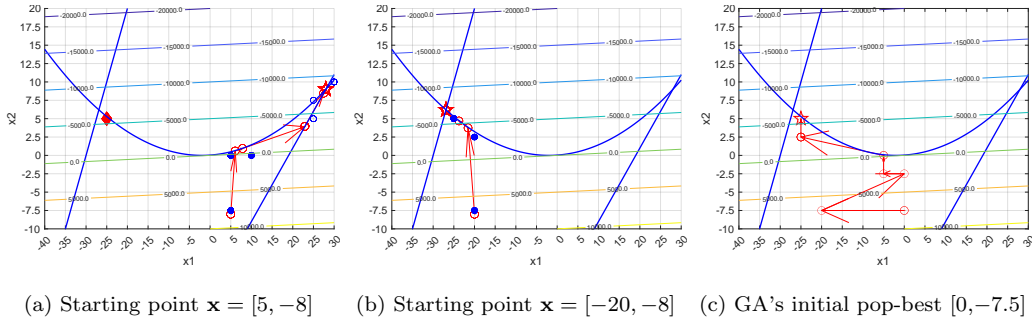


Figure 3: Iteration-wise intermediate points of Matlab's `fmincon()` (shown in (a) and (b) from different initial points) do not find the discrete optimal solution for the problem in Equation 4. However, a discrete GA finds the true discrete optimal solution $x^* = [-25, 5]$ having $f^* = -5,700$, shown in (c).

Figure 4 shows the result of a branch-and-bound algorithm, which is commonly and specifically designed to solve discrete optimization problems (1). It splits the original problem into two sub-problems using the location of the optimal solution of the continuous version $x^{*,\text{cont}} = [28, 9]$ and by adding an additional constraint ($x_1 \leq \lfloor 28_5 \rfloor = 25$ and ($x_1 \geq \lceil 28_5 \rceil = 30$) to the original problem. These two continuous versions (OPT2 and OPT3) are then solved using `fmincon()`, as shown in the variable space and in the

flowchart in Figures 4a and 4b, respectively. This process continues until a feasible solution is found (in OPT4, for example), an infeasible solution is found (OPT3, for example), or no node is open. Such a sophisticated and multiple sequential optimizations (OPT1 to OPT7) are needed to reliably find the true discrete optimal solution $\mathbf{x}^* = [-25, 5]$.

It is clear that an approximation of iteration-wise intermediate points of an optimization run on the continuous version of the problem, as shown in Figure 3, may not always discover the true discrete optimal solution. A more sophisticated optimization approach which handles discreteness of the variable space within the overall optimization process, and not as a post-optimal fix-up, is needed to find the desired solution.

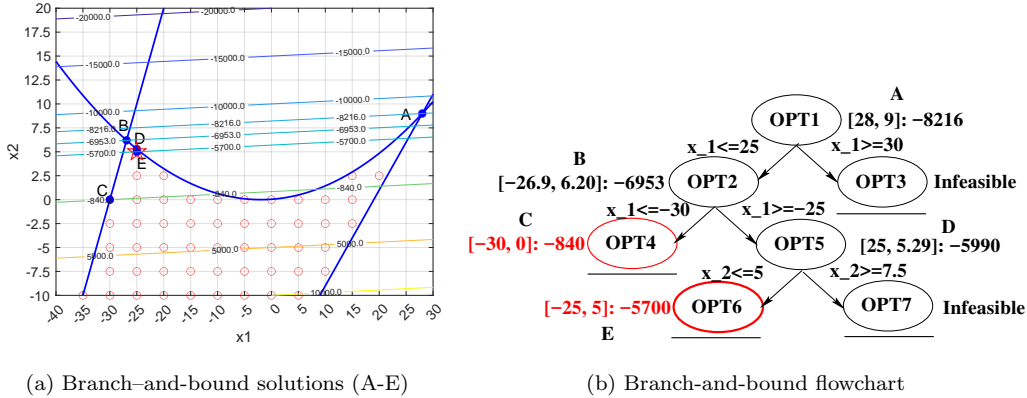


Figure 4: Allowed discrete variable vectors are marked with a red circle for the problem in Equation 4. Results of branch-and-bound method (in solid blue circles) show how the discrete optimal solution ($E=(-25, 5)$) is found starting from $A=(28, 9)$. Branching into six different sub-problems are needed to find the discrete optimal solution (E).

Finally, we show the result from a discrete GA, in which only allowable discrete values of each variable are permitted in its initial population and created offspring population. Real-coded GA with a population of 12 mem-

bers, a discrete SBX (30) ($p_c = 0.9$ and $\eta_c = 1$) and discrete polynomial mutation operators ($p_m = 0.5$ and $\eta_m = 5$) (31) finds the discrete optimal solution ($\mathbf{x}^* = [-25, 5]$) in only after seven generations. Since no approximation of variables from continuous to discrete values are used during the entire optimization process and only allowed discrete values are permitted, the optimization process is more likely to find the desired solution at the end.

5.3. Finding Multiple Optimal Solutions

To demonstrate the task of finding multiple optimal solutions for a multimodal problem, we use the famous Himmelblau problem having four different optimal solutions in the search space. Figure 5a shows the optimal solutions discovered by the Nelder-Mead algorithm of Matlab starting with four random initial points. First, the intermediate points of each search are never close to any of the four optimal solutions, except at the end. Second, two different optimization runs from different initial points end up close to one optimal solution, thereby not finding one of the four global optimal solutions in four independent runs.

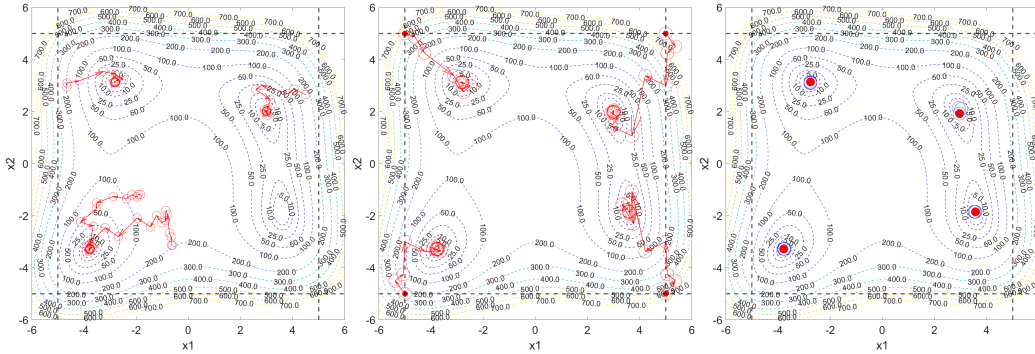
To find multiple optimal solutions systematically, we propose a method which starts with a distant initial point from the ones used before. For this purpose and for this problem, we choose a near-corner point in the search space as the first initial point. An optimization is run from this point and the optimal solution is saved. Next, a new initial point is chosen *furthest* from the first initial point and another optimization is performed from this initial point. The third optimization is executed from an initial point which is furthest away from both initial points of the earlier two optimization runs, and so on. As shown in Figure 5b, all four optimal solutions are now found by

this procedure. It is interesting to note the overall optimization algorithm is a sequence of four different and independent runs, each starting with a different initial point. Every run ends with a specific near-optimal solution at its final iteration and not obtained from any intermediate iteration. It is interesting to note that previous runs and their outcomes are used to create an initial point for the next optimization run. The overall algorithmic strategy does not use iteration-wise intermediate points in any way, but utilizes the outcome of runs to help focus the search in a different part of the search space. Such a strategy is acceptable as a reliable and repeatable optimization process.

To ensure finding multiple optima, a population-based optimization approach stays as a competing algorithm for parallelly searching in multiple optimal regions through its diverse population members. A niche-based GA (10; 11) is one such potential algorithm to find multiple optima in a single run. The results for a single run of a niche-based GA with 20 population members is run for 30 generations to solve the above problem. Figure 5c shows the final population at the end of the run has found all four optima (shown in red circles). This demonstrates again that if an algorithm is modified with the goal, the resulting algorithm may produce the desired outcome at the end.

5.4. Finding Multi-objective PO Solutions

We consider the following two-variable bi-objective problem to illustrate the outcome of various optimization algorithms issues related to the inclusion of iteration-wise intermediate points in determining the final solutions of the



(a) Nelder-Mead algorithm with (b) Nelder-Mead algorithm with (c) A niche-based GA finds all four four random starting points finds four adaptively generated starting global optima in a single run. three of the four global optima. points finds all four global optima.

Figure 5: Results on Himmelblau’s function having four optima: (a) random starting points find identical or distinct optimum, (b) a systematic distribution of initial points can find distinct optima in multiple optimization runs, and (c) finds all four optimal solutions in a single run using a population-based niched GA.

PF (3):

$$\begin{aligned}
 & \text{Minimize } f_1(\mathbf{x}) = x_1, \\
 & \text{Minimize } f_2(\mathbf{x}) = 1 + x_2^2 - x_1 - 0.1 \sin(3\pi x_1), \\
 & \text{subject to } 0 \leq x_1 \leq 1, \quad -2 \leq x_2 \leq 2.
 \end{aligned} \tag{5}$$

First, we use the weighted-sum approach (32) to convert the bi-objective problem into a parameterized single-objective optimization problem using a weight vector $\mathbf{w} = (w, 1 - w)$:

$$\begin{aligned}
 & \text{Minimize } f(\mathbf{x}) = w f_1(\mathbf{x}) + (1 - w) f_2(\mathbf{x}), \\
 & \text{subject to } 0 \leq x_1 \leq 1, \quad -2 \leq x_2 \leq 2.
 \end{aligned} \tag{6}$$

We choose $w = 0.35$ and apply Matlab’s `fmincon()` routine starting with an initial point $\mathbf{x}^{(0)} = (0.99, 1.00)$. Intermediate objective vectors are shown in Figure 6a. The objective space plot reveals that the initial and first in-

intermediate objective vectors are dominated by the final three objective vectors. Also, the algorithm quickly converges near the optimal solution of the weighted-sum problem.

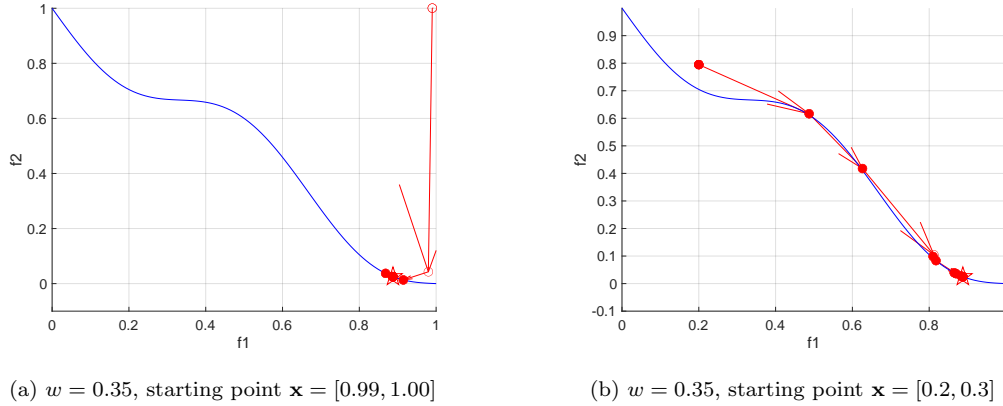


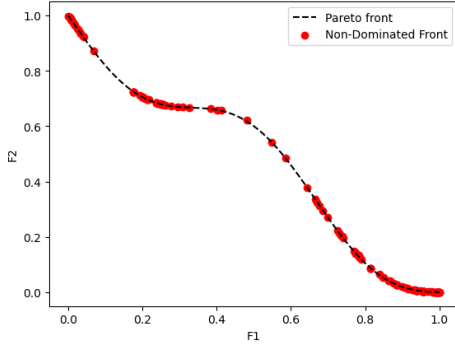
Figure 6: Iteration-wise intermediate points of Matlab’s `fmincon()` for different w may or may not find solutions on the non-convex part of the PF, depending on the chosen starting point and is entirely a happenstance, applied to the problem in Equation 5. Non-dominated intermediate points are marked in bold. In (b), the intermediate points seem to have found a few non-convex PO solutions, but they are not expected as a result of the weighted-sum approach.

Next, we use the same w value, but use a different initial point $\mathbf{x}^{(0)} = (0.2, 0.3)$. Figure 6b shows a remarkable result. All intermediate points are non-dominated to each other and some of the intermediate points fall on the non-convex part of the PF. However, the final solution (shown with a star) is the same desired PO solution as that obtained in Figure 6a. The fact that this run has picked a number of intermediate points on the non-convex part of the PF does not qualify it to be a legitimate generic multi-objective optimization algorithm capable of finding non-convex PO solutions. Theoretically and contrary to an earlier study (33), the weighted-sum method cannot

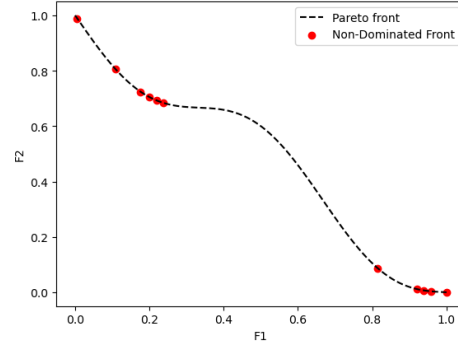
find any non-convex PO solution, no matter what non-negative weights are used. We argue here that the intermediate points are just stepping stones for the weighted-sum algorithm to approach to the final desired solution, which can only lie on the convex part of the PF. To select iteration-wise intermediate points and claim that the theoretical result associated with the weighted-sum approach, or even claim that the weighted-sum approach can be used to reliably find the non-convex part of the PF, is wrong. For the specific problem and specific initial point shown in this example, it was a happenstance and as shown in Figure 6a this may not happen every time.

To make the matters more profound, let us execute Matlab's `fmincon()` 11 times from different initial points, collect all iteration-wise intermediate points, and investigate the non-dominated solutions. Figure 7a shows these solutions on the objective space. It is clear that a nicely distributed set of PO solutions on the entire PF is found. Some of the solutions came from the intermediate iterations and some may have come from the final iteration, but together they represent a well-distributed PF, violating the theoretical understanding of the weighted-sum approach. By considering iteration-wise intermediate points can sometimes result in a false pretense of the outcome of an optimization algorithm. However, when only the final solution of each of the 11 runs are plotted together in Figure 7b, there is no solution found to lie on the non-convex part of the PF. This is the correct behavior of the weighted-sum algorithm, supporting the theoretical understanding. By using only the final solutions as outcomes of an algorithm, the true nature of an optimization algorithm can be understood.

Another reliable way to find a well-distributed set of PO solutions is to



(a) Non-dominated solutions of combined *intermediate* solutions of 11 runs of `fmincon()`.



(b) Non-dominated solutions of combined *final* solutions of 11 runs of `fmincon()`.

Figure 7: Non-dominated solutions from a collection of all (a) iteration-wise intermediate points and (b) final solutions from 11 runs of Matlab’s `fmincon()` indicate the true property of the weighted-sum approach revealed by (b) for the problem in Equation 5.

make sure that the optimization algorithm is specifically designed for this task. Evolutionary multi-objective optimization (EMO) algorithms are designed for two goals in mind: (i) convergence to the PO set and (ii) maintain a widely distributed set of solutions. We employ the NSGA-II procedure (12) with a population size of 100 and run for 100 generations to obtain the non-dominated solutions plotted (Figure 8). Irrespective of convexity of the PO set, NSGA-II and other EMO algorithms can find a widely distributed set of near PO solutions and capture them in a population, so the algorithm does not have to be applied multiple times.

5.5. Finding Innovation Path Solutions

Recently, authors have introduced an Innovation Path (IP) problem and proposed a bi-objective IP-seeking algorithm (34) to find a set of optimal intermediate *anchor* solutions from a given current solution \mathbf{x}^C toward a

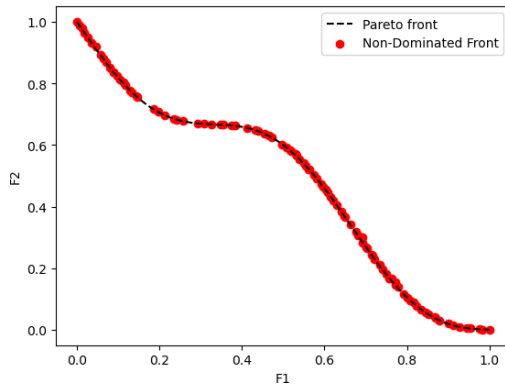


Figure 8: A widely distributed set of non-dominated solutions at the final generation of an NSGA-II run for the problem in Equation 5.

desired target \mathbf{x}^T . The method uses the ”multiobjectivization” principle proposed by EMO researchers in the past (35; 36; 37; 38; 3; 1). The proposed EMO-based IP-seeking algorithm is sophisticated and makes an optimal trade-off between an improvement in the original objective ($f(\mathbf{x})$) and the Euclidean difference from the current solution \mathbf{x}^C :

$$f_D(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}^C\|_2. \quad (7)$$

using step-constraints defined using any solution \mathbf{x} and the previous anchor solution $\mathbf{x}^{(k)}$:

$$\widehat{G}_l(\mathbf{x}, \mathbf{x}^{(k)}) \leq 0, \quad l = 1, 2, \dots, L. \quad (8)$$

The IP-seeking method has been modified for the improvement of multiple objective simultaneously (34). After the IP-seeking optimization algorithm is performed, it results in a sequence of anchor solutions, which should not be confused with iteration-wise intermediate points of a typical optimization algorithm. However, the task in the IP-seeking algorithm may get confused with a task in a typical single-objective optimization.

Let us consider the minimization of the Himmelblau’s function in which the search space is the first coordinate ($x_1 \geq 0, x_2 \geq 0$). If we start with an initial point $x^{(0)} = (5, 5)$, we can apply an optimization algorithm to find the minimum ($\mathbf{x}^* = (3, 2)$). Matlab’s `fminsearch()` algorithm shows the iteration-wise intermediate points in Figure 9a in red circles. The intermediate points are not supposed to follow any specific trade-off among certain goals. The intermediate points are outcomes of the update principles of the optimization algorithm laid out at the time of designing the algorithm. As long as the intermediate points finally lead to the near the optimal solution, the purpose of the optimization process is served. However, in IP-seeking algorithm, we would like the initial solution (known as the current solution) must transform to the optimal solution in a path which makes an optimal trade-off between the improvement in $f(\mathbf{x})$ value and difference from the current point in a few steps, so that transformation is practically viable. For the sake of plotting, we mark the non-dominated intermediate points with filled circles and dominated ones in open circles.

When the IP-seeking algorithm is applied to the same problem, starting with the same initial point as the current point and a step constraint $G_1(\mathbf{x}, \mathbf{x}^{(k)}) \equiv 1 - (f_D(\mathbf{x}) - f_D(\mathbf{x}^{(k)})) \leq 0$, four anchor points are found, shown in filled blue circles. First, notice that the IP-anchor solutions are more organized. Second, notice when these points are plotted in the two-objective space (Figure 9b), the IP-anchor solution set dominates mostly the `fminsearch()`-obtained intermediate points. This clearly states that applying a typical optimization algorithm and choose the non-dominated intermediate points may not produce certain desired trade-off among the solutions.

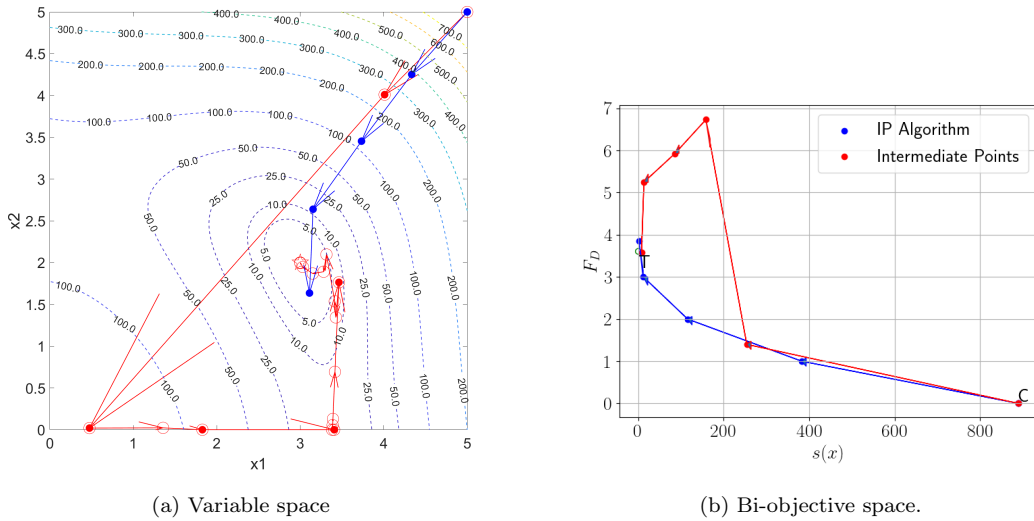


Figure 9: Intermediate points of an optimization algorithm (red) and anchors of an IP-seeking algorithm (blue) in (a) variable space and (b) objective space, for the IP problem constructed with the Himmelblau function. Intermediate points are inferior to IP-anchors in the context of trade-off between improvement in $f(\mathbf{x}) = s(\mathbf{x})$ and difference ($f_D(\mathbf{x}) = F_D(\mathbf{x})$) from the current point.

What is desired must be somehow implemented inside an algorithm as a goal. Not always, looking for what is desired within the intermediate points of an optimization run will allow to achieve the goal.

5.6. Engineering Optimization Problems

Finally, we demonstrate the drawbacks of exploiting iteration-wise intermediate points from typical optimization runs for three different engineering optimization tasks.

5.6.1. Robust Optimization of Welded-beam Design Problem

We start by finding the robust solution for the single objective (cost) welded-beam design problem (1). Table 1 shows the results of different meth-

ods and the optimal/robust solutions found. First, the minimal solution of the original problem (without any robustness consideration) is obtained using Matlab’s `fmincon()`’s SQP algorithm and is presented in the first row. It is the best-known solution reported in the literature (1). The constraint violation (CV) for this solution is zero, indicating that it satisfies all four constraints of the problem. For this solution, the $R(\mathbf{x})$ column indicates the robustness measure (Equation 2) with $r = 1000$. A total of $H = 25$ random points are computed in the neighborhood (in $[-0.05, 0.05]$ of every x_i) of this minimal solution, making a total of 26 points to calculate the robustness metric. It is observed that 24 out of 26 neighboring solutions (including itself) are infeasible, thereby making $R(\mathbf{x})$ value much larger than $f(\mathbf{x})$ and the total normalized constraint violation is 6.5759.

Table 1: Optimal and robust solutions for the welded-beam problem using different optimization methods. $CV(\mathbf{x})$ is the normalized constraint violation of \mathbf{x} . I denotes the number of infeasible points out of 26 solutions in the neighborhood of \mathbf{x} , and $Tf(\mathbf{x})$ and $TCV(\mathbf{x})$ are the total f -values and total normalized constraint violation from all neighboring solutions including \mathbf{x} .

Algorithm	x_1	x_2	x_3	x_4	$f(\mathbf{x})$	$CV(\mathbf{x})$	$R(\mathbf{x})$	I	$Tf(\mathbf{x})$	$TCV(\mathbf{x})$
Original SQP	0.2444	6.2175	8.2915	0.2444	2.381	0.0	255.290	24	61.5	6.5759
Robust Interm.	2.5625	5.0500	5.0500	2.5625	48.493	0.0	51.748	10	1259.7	0.0857
Robust SQP	0.7635	2.8897	4.6007	0.8469	5.027	0.0	5.096	2	130.4	0.0021
Robust GA	0.2813	7.8698	7.2865	0.3641	3.480	0.0	3.475	1	90.1	0.0003

Next, all intermediate points of the above optimization run is stored. For each intermediate point, the robustness measure is computed with the same neighborhood framework. The minimum $R(\mathbf{x})$ solution of all intermediate

points is presented in the second row. Note that the $R(\mathbf{x})$ value is much smaller than that from the minimal solution, indicating that this solution is more robust according to our metric than the minimal solution; however, the objective function values of this solution and its neighboring solutions are much inferior, indicated by a large Tf value. Importantly, this solution lies in the search space having 10 of 26 neighboring solutions being infeasible, but the overall constraint violation is much smaller than the minimal solution, as indicated by a TCV value of 0.0857. This indicates that by saving iteration-wise intermediate points and choosing the minimum robustness metric value solution may apparently find a better robust solution than the minimal solution, but the following study indicates that it is not even close to the true robust solution.

To obtain the true robust solution, we now minimize the robustness metric $R(\mathbf{x})$, instead of minimizing the original objective function $f(\mathbf{x})$. Row 3 shows the best robust solution found by simply minimizing the $R(\mathbf{x})$ function using the SQP algorithm of `fmincon()`, presented in Equation 2. It is clear that it is a feasible solution (as $CV(\mathbf{x}) = 0$) and the $R(\mathbf{x})$ value is also much smaller than that of the earlier two solutions. Due to the inclusion of the constraint violation term with $r = 1000$ in $R(\mathbf{x})$, this solution lies in the search space having only 2 out of 26 solutions infeasible in its vicinity with a TCV of 0.0021. A comparison of the variable vectors Rows 2 and 3 indicates that they are quite different from each other. Although this robust solution has a slightly larger objective value ($f(\mathbf{x})$) than the original minimum solution ($f = 2.381$), it is definitely more robust, as indicated by the respective $R(\mathbf{x})$ values. Interestingly, this solution is also more robust

than the robust intermediate solution (second row). Such a solution is not possible without directly minimizing a metric which represents the sensitivity of the solution in its neighborhood.

Finally, a population-based real-parameter GA is run to minimize the robustness function and Row 4 shows the resulting solution. The GA solution is slightly more robust than the SQP-obtained robust solution (Row 3), and it also has slightly worse f value than that in Row 3. It causes only 1 out of 26 neighboring solutions infeasible with a TCV value of 0.0003. The GA robust solution is also quite different from the robust SQP solution, but interestingly it is a small variation of the minimal solution of $f(\mathbf{x})$, shown in Row 1. Rge iteration-wise intermediate solutions of original SQP minimization of $f(\mathbf{x})$ did not go close to this robust GA solution and failed to have such a point in its intermediate point set. Although both Robust SQP and GA optimize the same robustness function, the complicated nature of $R(\mathbf{x})$ causes the SQP solution to get stuck at a non-optimal solution. Nevertheless, these two robust optimization algorithms are able to find better robust solutions in the search space with smaller R -value and fewer infeasible neighboring solutions than the first two solutions which did not truly optimize any robustness measure in their optimization or selection process.

This example illustrates that if robust solutions are to be found, an appropriate objective function implementing a robustness measure must be chosen. Solving the original problem and looking for a robust solution in its iteration-wise intermediate solution set may not lead to the desired solution.

5.6.2. Bi-objective Optimization for Crash-worthiness Problem

Next, we use the two-objective (minimizing mass and acceleration) of the crash-worthiness problem (39) to demonstrate the effect of various optimization algorithms on a non-convex problem. Figure 10 shows the intermediate points discovered by two separate runs of the Matlab's `fmincon()` routine with different initial points but having the same weight vector $\mathbf{w} = (0.25, 0.75)$ for the weighted-sum approach. The PF found by NSGA-II with standard parameter setting (12) is shown with blue circles. The non-convexity of the PF is clear from the figure. As seen, Run 1 discovers some non-dominated points (marked in filled red circles) that are close to or on the non-convex part of the PF. However, it should not be surmised that the weighted-sum approach can find non-convex PO solutions. The final solution found by Run 1 is on the PF (the right extreme point) and is expected. Run 2, starting with a different initial point takes a different path to arrive at the identical PO point to that obtained by Run 1. All intermediate points are dominated by the final point. Only when an optimization algorithm is designed to handle multiple conflicting objectives by emphasizing non-dominated and diverse solutions with a population-based framework, as in NSGA-II, multiple PO solutions can be found in a single simulation run. Looking for intermediate points to find non-dominated and diverse solutions is not expected to provide PO solutions reliably.

5.6.3. Innovation Path Task for Welded-beam Design Problem

Finally, we consider the two-objective version of the welded-beam design problem (1), but now investigate the task of finding optimally a set of trade-off IP-anchors from the current solution ($\mathbf{x}^C = (2, 4, 6, 2)$) having

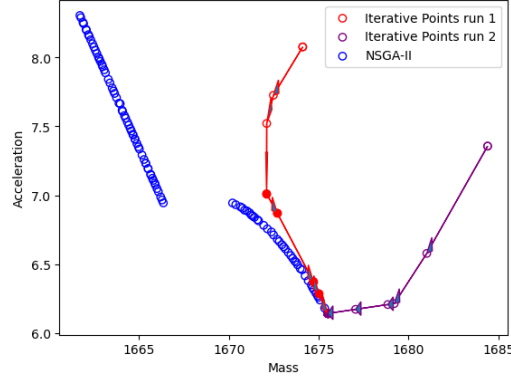


Figure 10: Iteration-wise intermediate points discovered by Matlab’s `fmincon()` from two different starting points to solve the same weighted-sum problem with a weight vector $(0.25, 0.75)$ and the PF discovered by NSGA-II (blue circles) for the crash-worthiness problem having two objectives.

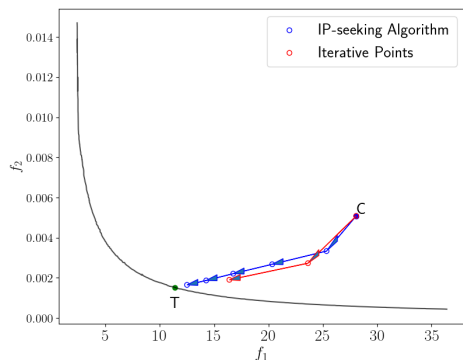
$f(\mathbf{x}^C) = (28.07, 0.0051)$ toward a target solution. The optimal solution of an achievement scalarization function (ASF) (40) is the target solution for our illustration:

$$s(\mathbf{x}) = \max_{i=1}^2 \frac{\hat{f}_i(\mathbf{x}) - \hat{z}_i}{w_i}, \quad (9)$$

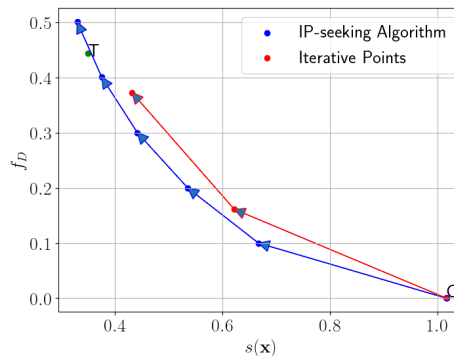
with $\mathbf{w} = (0.95, 0.05)$ and $\hat{\mathbf{z}} = (0, 0)$.

Figure 11 shows the results from two approaches. The IP-anchors (blue points) found by the IP-seeking algorithm (41) using a step-constraint $G_1(\mathbf{x}, \mathbf{x}^{(k)}) \equiv 0.1 - (f_D(\mathbf{x}) - f_D(\mathbf{x}^{(k)})) \leq 0$, in the bi-objective space. The iteration-wise intermediate points found by Matlab’s `fmincon()` routine applied to solve the above ASF problem, started from the current solution, are marked in red circles. Apparently, both algorithms have found a few similar solutions toward the target, but as shown in Figure 11b, IP-anchors (blue circles) *dominate* the iteration-wise intermediate points obtained by the minimization of $s(\mathbf{x})$. Additionally, the IP-seeking approach discovers a better uniformly-

distributed anchors as specified by the step constraints.



(a) Anchors found by both iteration-wise and IP seeking algorithm in the original search space



(b) Anchors for both algorithms in the bi-objective problem space

Figure 11: intermediate points of Iteration-wise Matlab's (`fmincon()`) and IP-seeking algorithm in original search space (a), bi-objective problem space (b) for the welded beam problem.

This example illustrates that if a sequence of trade-off IP-anchors need to be found from a starting solution, the ensuing optimization algorithm must be modified to search for trade-off solutions. Just minimizing the objective function and then looking for non-dominated intermediate solutions is not expected to provide desired solutions.

6. Acceptable and Unacceptable Practices

Based on the above discussions and results, we discuss here some unacceptable and acceptable practices of optimization methodologies.

6.1. Unacceptable Practices

It is clear that iteration-wise intermediate points should not be used as final outcome of an optimization algorithm. They are merely stepping-stones

toward the final solution. Any iterative optimization algorithm starts with one or more initial points and ends up with one or more final optimized solutions as an outcome. The path from initial to the final iteration depends intricately on the chosen algorithm and should not be given any more importance as much as possible, due to the number of reasons outlined in Section 4. This fact is also demonstrated through a series of case studies in the previous section. For the same reasons, the external archive based EMO studies (8; 9), which stores an updated list of all non-dominated solutions from start to end and then pick a few through a subset selection process using a multi-objective criterion, must be completely avoided. Allowing this practice to be a norm may lead researchers to develop different un-intelligent algorithms which may rely on scavenging the search space for any solution and storing them in the growing external archive, knowing fully that the archive will be its savior. Such a process will inhibit the progress toward developing intelligent optimization algorithms which so carefully compare current and previous iteration of solutions to decide which solutions need to be promoted to meet the goals of the optimization task and importantly achieve the task repeatedly with less computational effort and with solid fundamental understanding supported by sound explanation of its working principles.

6.2. Acceptable Practices

Having said that, let us now discuss what could be acceptable as a reliable, repeatable, and respectful optimization algorithm. Clearly, it is the final solution or solution set which should be utilized as the outcome of an optimization algorithm. However, there can be some ways to utilize the

iteration-wise intermediate points, not as a final outcome, but to improve an algorithm's performance. We highlight some such acceptable practices in the following paragraphs.

6.2.1. "Best of Multiple Runs" Approach

Many applied optimization studies apply an optimization algorithm multiple times starting from different initial populations and the best solution of final outcome of individual runs is declared as the final optimized solution. Although such an approach is associated with more solution evaluations to complete multiple optimization runs, if affordable, such a method is an acceptable optimization methodology, as each run's final optimized solution is considered as the final outcome. For this purpose, keeping the algorithm unchanged, each optimization run can be initiated with a point or a population, which is derived by some logic and away from the previously chosen initial point(s) and the respective obtained final outcome. In Section 5.3, we have proposed such an adaptive multi-run optimization method to find multiple optimal solutions, as a viable and reliable approach. However, combining iteration-wise intermediate points from one or multiple independent runs and determining the final outcome of the overall procedure may not be acceptable as a reliable strategy.

6.2.2. Multi-stage Approaches

First, the iteration-wise intermediate points can be stored and analyzed at the end of an optimization run to select good seed solutions for the next-stage of an optimization run. As if, the first optimization run which started with a random initial point or population, was considered difficult for the

algorithm to find near-optimal solutions and the algorithm is purposefully used to create one or more new initial points for the next-stage algorithm. Thus, we argue that the multi-stage optimization algorithms, proposed in the single and multi-objective optimization literature (42; 43), are acceptable approaches, as each stage completes an optimization process and the outcome of the previous stage is used as starting points for the next stage. The final solutions of the final stage of optimization algorithm are then used as the outcome of the overall procedure. This approach is perfectly legitimate.

6.2.3. Restart Approaches

In this sense, *restarts* (44; 45), often used in the optimization literature, are acceptable approaches as well. Restart methods start the same optimization algorithm by using the information of the final solutions where the algorithm got stuck in the previous stage. The final solutions of the overall multiple restart-based optimization method is declared as the final outcome of the procedure.

6.2.4. Elite-preserving Approaches

This brings us to the elite-preserving optimization algorithms (12; 46) often used in the evolutionary optimization literature. In these algorithms, at each iteration, the best or a few best solutions from the previous iteration is/are compared with the current population and the best solutions are carried forward to the next iteration. An operator to achieve this task is called a survival selection operator. In some sense, these strategies can be considered as algorithms with restarts happening at each iteration by choosing the best solutions from previous iteration as parts of new starting population.

This way, these algorithms always produce a non-deteriorating performance. The algorithm judiciously decides which points to carry on from iteration to iteration to arrive at the final optimized solutions. According to our argument, such methodologies are acceptable, as long as the final solution of the process is declared as the outcome of the overall optimization process. They are different from passively saving all iteration-wise intermediate points in an archive and at the end pick the most suitable ones based on a metric as an outcome.

7. Conclusions

This paper has criticized the use of iteration-wise intermediate points as possible outcome of a reliable, repeatable, and intelligent optimization algorithm. The main reasons against their use are (i) intermediate points are never the final targets of an optimization algorithm, rather they are stepping-stones in a sequence of points that eventually lead close to the optimal solutions, (ii) intermediate points intricately depend on the initial points an algorithm was started with and hence may not be repeatable, (iii) expecting to find the desired solutions in a set of non-targeted intermediate solutions is purely relying on chance, and (iv) making the scavenging process to discover desired solutions from intermediate solutions prominent for an optimization algorithm does not make the process intelligent and respectful.

To illustrate, we have discussed and presented results on several typical optimization tasks seeking (i) robust solutions, (ii) discrete solutions, (iii) multiple optimal solutions in a multi-modal problem, (iv) multiple Pareto-optimal (PO) solutions, and (v) a sequence of innovation path solutions. In

each case, we have argued reasons why iteration-wise intermediate points should not be used to select the outcome of the optimization process. Although in some problems, they may appear to be close to a desired solution, but when this happens, it is a mere chance and may not be repeatable. Only by considering the final solutions as outcome of an optimization algorithm can demonstrate purpose, consistency, repeatability and reliability of optimization algorithms. More such illustrations can be made to further highlight this aspect, but various optimization tasks illustrated here clearly indicate that an optimization algorithm must be infused with the goal to directly and more reliably find desired solutions, rather than foraging one or more of the intermediate solutions as a post-optimal task as the outcome. Finally, we have argued some acceptable extensions of a standard optimization algorithm which may utilize the iteration-wise intermediate points to gather knowledge to improve an algorithm's starting conditions for a further application, but expecting to achieve the desired goals directly from the iteration-wise intermediate points from an optimization run is too ambitious, relies on happenstance, and may not be repeatable, and therefore, should not be practiced.

References

- [1] G. V. Reklaitis, A. Ravindran, K. M. Ragsdell, Engineering Optimization Methods and Applications, New York : Wiley, 1983.
- [2] K. Deb, Optimization for Engineering Design: Algorithms and Examples, New Delhi: Prentice-Hall, 1995.

- [3] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*, Wiley, Chichester, UK, 2001.
- [4] M. S. Bazaraa, H. D. Sherali, C. M. Shetty, *Nonlinear programming: Theory and algorithms*, Singapore: Wiley, 2004.
- [5] J. H. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: MIT Press, 1975.
- [6] D. E. Goldberg, *Genetic Algorithms for Search, Optimization, and Machine Learning*, Reading, MA: Addison-Wesley, 1989.
- [7] C. A. C. Coello, D. A. VanVeldhuizen, G. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*, Boston, MA: Kluwer, 2002.
- [8] M. Li, X. Yao, Quality evaluation of solution sets in multiobjective optimisation: A survey, *ACM Computing Surveys* 52 (2) (2019) 1–38.
- [9] H. Ishibuchi, L. M. Pang, K. Shang, A new framework of evolutionary multi-objective algorithms with an unbounded external archive, *Authorea Preprints* (2023).
- [10] D. E. Goldberg, J. Richardson, Genetic algorithms with sharing for multimodal function optimization, in: *Proc. of the First Int. Conf. on Genetic Algorithms and Their Applications*, 1987, pp. 41–49.
- [11] K. Deb, D. E. Goldberg, An investigation of niche and species formation in genetic function optimization, in: *Proceedings of the Third International Conference on Genetic Algorithms*, 1989, pp. 42–50.

- [12] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast and elitist multi-objective genetic algorithm: NSGA-II, *IEEE Transactions on Evolutionary Computation* 6 (2) (2002) 182–197.
- [13] Q. Zhang, H. Li, MOEA/D: A multiobjective evolutionary algorithm based on decomposition, *Evolutionary Computation, IEEE Transactions on* 11 (6) (2007) 712–731.
- [14] K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point based non-dominated sorting approach, Part I: Solving problems with box constraints, *IEEE Transactions on Evolutionary Computation* 18 (4) (2014) 577–601.
- [15] H. Jain, K. Deb, An evolutionary many-objective optimization algorithm using reference-point based non-dominated sorting approach, Part II: Handling constraints and extending to an adaptive approach, *IEEE Transactions on Evolutionary Computation* 18 (4) (2014) 602–622.
- [16] A. Alvarado, G. Scutari, J.-S. Pang, A new decomposition method for multiuser dc-programming and its applications, *IEEE Transactions on Signal Processing* 62 (11) (2014) 2984–2998.
- [17] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, *IEEE Transactions on Evolutionary Computation* 1 (1) (1997) 67–82.
- [18] D. W. Corne, J. D. Knowles, No free lunch and free leftovers theorems for multiobjective optimisation problems, in: *Proceedings of the 2nd international conference on Evolutionary multi-criterion optimization, EMO'03*, Springer-Verlag, Berlin, Heidelberg, 2003, pp. 327–341.

- [19] K. Deb, U. B. Rao, S. Karthik, Dynamic multi-objective optimization and decision-making using modified NSGA-II: A case study on hydro-thermal power scheduling bi-objective optimization problems, in: Proceedings of the Fourth International Conference on Evolutionary Multi-Criterion Optimization (EMO-2007), 2007, pp. 803–817.
- [20] Y. Jin, B. Sendoff, Constructing dynamic optimization test problems using the multiobjective optimization concept, in: Lecture Notes in Computer Science, 3005, 2004, pp. 525–536.
- [21] C. Raquel, X. Yao, Dynamic multi-objective optimization: a survey of the state-of-the-art, in: Evolutionary Computation for Dynamic Optimization Problems, Springer, 2013, pp. 85–106.
- [22] A. Sinha, P. Malo, K. Deb, A review on bilevel optimization: From classical to evolutionary approaches and applications, IEEE Transactions on Evolutionary Computation 22 (2) (2018) 276–295.
- [23] J.-A. Mejía-de Dios, E. Mezura-Montes, A physics-inspired algorithm for bilevel optimization, in: 2018 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC), 2018, pp. 1–6. [doi:10.1109/ROPEC.2018.8661368](https://doi.org/10.1109/ROPEC.2018.8661368).
- [24] Z. Lu, K. Deb, E. Goodman, J. Wassick, Solving a supply-chain management problem using a bilevel approach, in: GECCO '17: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, New York, NY, USA, 2017, pp. 1185–1192.
- [25] K. Deb, S. Tiwari, Omni-optimizer: A generic evolutionary algorithm for

- global optimization, *European Journal of Operations Research (EJOR)* 185 (3) (2008) 1062–1087.
- [26] K. Deb, An efficient constraint handling method for genetic algorithms, *Computer Methods in Applied Mechanics and Engineering* 186 (2–4) (2000) 311–338.
- [27] K. Deb, H. Gupta, Introducing robustness in multi-objective optimization, *Evolutionary Computation Journal* 14 (4) (2006) 463–494.
- [28] K. Deb, S. Gupta, D. Daum, J. Branke, A. Mall, D. Padmanabhan, Reliability-based optimization using evolutionary algorithms, *IEEE Trans. on Evolutionary Computation* 13 (5) (2009) 1054–1074.
- [29] A. Gaspar-Cunha, J. A. Covas, Robustness in multi-objective optimization using evolutionary algorithms, *Computational Optimization and Applications* 39 (2008) 75–96.
- [30] K. Deb, R. B. Agrawal, Simulated binary crossover for continuous search space, *Complex Systems* 9 (2) (1995) 115–148.
- [31] K. Deb, D. Deb, Analysing mutation schemes for real-parameter genetic algorithms, *International Journal of Artificial Intelligence and Soft Computing* 4 (1) (2014) 1–28. [doi:DOI:10.1504/IJAISC.2014.059280](https://doi.org/10.1504/IJAISC.2014.059280).
- [32] K. Miettinen, *Nonlinear Multiobjective Optimization*, Kluwer, Boston, 1999.
- [33] Y. Jin, M. Olhofer, B. Sendhoff, Dynamic weighted aggregation for evolutionary multi-objective optimization: Why does it work and how, in: *Proceedings of the genetic and evolutionary computation conference, 2001*, pp. 1042–1049.

- [34] A.Khan, K.Deb., [Innovation path: Discovering an ordered set of optimized intermediate solutions from an existing to a desired solution](#), in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 529–537. URL <https://doi.org/10.1145/3638529.3654051>
- [35] J. Branke, K. Deb, K. Miettinen, R. Slowinski, Multiobjective optimization: Interactive and evolutionary approaches, Springer-Verlag, Berlin, Germany, 2008.
- [36] J. D. Knowles, D. W. Corne, K. Deb, Multiobjective problem solving from nature, Springer Natural Computing Series, Springer-Verlag, 2008.
- [37] C. A. C. Coello, Treating objectives as constraints for single objective optimization, *Engineering Optimization* 32 (3) (2000) 275–308.
- [38] E. Mezura-Montes, C. A. C. Coello, Constraint-handling in nature-inspired numerical optimization: past, present and future, *Swarm and Evolutionary Computation* 1 (4) (2011) 173–194.
- [39] X. Liao, Q. Li, W. Zhang, X. Yang, Multiobjective optimization for crash safety design of vehicle using stepwise regression model, *Structural and Multidisc. Optimization* 35 (2008) 561–569.
- [40] A. P. Wierzbicki, The use of reference objectives in multiobjective optimization, in: G. Fandel, T. Gal (Eds.), *Multiple Criteria Decision Making Theory and Applications*, Berlin: Springer-Verlag, 1980, pp. 468–486.
- [41] A. Khan, K. Deb, Towards an efficient innovation path seeking algorithm using directed domination, in: *Evolutionary Multi-Criterion Optimization*, Springer Nature Singapore, Singapore, 2025, pp. 3–16.

- [42] J. Cabello, J. Cejudo, M. Luque, F. Ruiz, K. Deb, R. Tewari, Optimization of the size of a solar thermal electricity plant by means of gas, *Renewable Energy* 36 (11) (2011) 3146–3153.
- [43] K. Deb, C. L. do Val Lopes, F. V. C. Martins, E. F. Wanner, Identifying pareto fronts reliably using a multistage reference-vector-based framework, *IEEE Transactions on Evolutionary Computation* 28 (1) (2023) 252–266.
- [44] A. S. Fukunaga, Restart scheduling for genetic algorithms, in: *International Conference on Parallel Problem Solving from Nature*, Springer, 1998, pp. 357–366.
- [45] D. Kim, J. A. Fessler, Adaptive restart of the optimized gradient method for convex optimization, *Journal of Optimization Theory and Applications* 178 (1) (2018) 240–263.
- [46] S. Sun, Y. Jiang, J. Wang, Automatic design of pixelated near-zero refractive index metamaterials based on elite-preserving genetic algorithm optimization, *Results in Physics* 48 (2023) 106461.