Unorthodox Optimization Tasks for Achieving Well-Informed Design for an Automobile Industry

Abhinav Gaur, AKM Khaled Talukder, Kalyanmoy Deb,
Santosh Tiwari, Simon Xu and Don Jones[♯]

COIN Report Number 2018005

Abstract

Engineering optimization problems from practice come with a number of complexities, which make generic and pedagogical optimization algorithms difficult to handle them in a straightforward manner. First, the problems usually involve a large number of variables and constraints. Second, the optimization task must, most often, be performed for a number of conflicting objectives. Third, practitioners need to find a near-optimal or a better-than-existing solution quickly, despite the fact that solution evaluations can be computationally expensive. Fourth, a curious practitioner is also often interested in not one, but multiple diverse yet equally good solutions, in order to compare a set of choices before selecting a single solution. Fifth, before the final acceptance of a preferred solution, they are often interested in performing a sensitivity analysis around the chosen solution, so that they are satisfied with any trade-off advantage which a neighboring solution may provide. Sensitivity is usually considered from a number of different issues, such as uncertainty in parameters and variables in implementing the solution, noise in objective and constraint functions, discrepancies between simulation models for objectives and constraints and actuality, etc. In this paper, the sensitivity of a solution is considered from the point of constraint violation. It helps the practitioners to know the maximum possible gain in critical objectives for a conceivable relaxation of the original constraints. Advanced optimization methods are used for meeting some of the above practicalities through a large-scale engineering design problem obtained from an automobile industry. Such approaches allow industrial practitioners to not only have an optimized solution at the end, but also to gather useful knowledge about the problem, which will help them lead in their profession and use the knowledge for future applications. Methods suggested in this paper are generic and can be applied to other similar engineering optimization problems.

1 Introduction

Traditionally, optimization problems must be formulated in a certain structured manner before a suitable optimization algorithm can be applied to find respective optimal solutions. But practice may demand a different description of the problem to achieve a desired property of the resulting optimized solutions, which may not fit with the existing structured optimization formulations. The rigidity of conventional optimization procedures is one of the reasons for certain apathy of industries to adopt optimization methods in their routine applications.

However, in the recent past, new and flexible optimization algorithms have been proposed. They are already capable of handling a number of practicalities: (i) discrete and mixed-integer nature of the search space, (ii) a large number of variables, (iii) multiple conflicting objectives, (iv) a large number of constraints, (v) black-box nature of objective and constraint evaluations, (vi) computationally expensive evaluation models, (vii) a small budget of solution evaluations, (viii) dynamic nature of objectives and constraints, (ix) uncertainties in problem parameters and variables, (x) hierarchical nature of variables, objectives and constraints, and others. These practicalities demand an unconventional optimization procedure to be

*A. Gaur, A.K.A Talukder and K. Deb are with Computational Optimization and Innovation (COIN) Laboratory, Michigan State University, East Lansing, MI, 48824 USA. e-mail: {gaurabhi, talukde1, kdeb}@msu.edu

†S. Tiwari, S. Xu and D. Jones are with the General Motors Vehicle Optimization Group, 30001 Van Dyke, Warren, MI 48093 USA. e-mail: {santosh.tiwari, simon.xu, don.jones}@gm.com

flexible, adaptable and efficient. The rise of population-based and flexible optimization algorithms over the past two decades has facilitated handling of above tasks in a reliable and routine manner.

Even after handling these practicalities by tractable methods, the practitioners often demand additional tasks which are expected to provide them valuable and well-informed solutions, before they accept a single preferred solution at the end of the design optimization process. Some of these additional tasks which are not often discussed in traditional academic optimization research, but are valuable in practice and are considered here, are: (i) optimization for multiple conflicting objectives, but having a keen interest in optimizing a single specific objective, (ii) identification of multiple diverse solutions having similar high-quality objective values, so that a single preferred solution can be chosen with a well-informed design activity, and (iii) searching around a chosen solution for a trade-off between constraint relaxation and objective improvement, so that a final negotiation with various disciplines can be initiated to revise the constraint bounds for a better objective solution. None of these tasks fall into a conventional optimization task, so that a standard methodology can be adopted in a straightforward manner. These deviations from conventional optimization practices are desired so that designers can be well-informed about other potential alternate solutions, before choosing a single, well-informed and well-evaluated design.

Optimization methods can be broadly classified into two classes: (i) a *point-based* method in which an algorithm uses a single point in each iteration to constitute its search process and ends up finding a single optimized solution, and (ii) a *population-based* method in which a number of points are used in each iteration and is capable of finding one or more optimized solutions. Both classes of algorithms have their merits and demerits. The point-based methods are usually faster and are often based on an optimality theory, but are usually less flexible for an update to handle practicalities mentioned above. On the other hand, population-based methods establish an implicitly parallel search process thereby making an efficient and fast identification of promising regions of the search space; they are easily parallelizable, and they are flexible to adapted to handle various practicalities mentioned above. Usually, the population-based methods do not employ any optimality theoretic results, but can be implemented [6], if desired. For these reasons, point-based methods have been demonstrated to be excellent local search methods for exploring a relatively small neighborhood in the search space effectively. On the other hand, evolutionary population-based optimization (EO) methods have been shown to have a more global perspective and are apt in finding diverse and multiple high-performing solutions. Due to their direct approach, EO methods are ideal for black-box optimization. Thus, if a problem falls in the purview of a local search method, point-based methods are ideal, but for problems requiring a more complex and global search, population-based methods with and without a local search method are ideal.

While an optimization algorithm is expected to find the true optimum solution of a problem, it often ends up finding a near-optinal solution, mainly due to the limited number of solution evaluations that are allowed to achieve within a stipulated amount of computational time allocated to solve the problem. Thus, in solving such real-world industrial problems, the goal is usually to find a near-optimum solution with as low a budget of solution evaluations as possible. In addition to finding a near-optimal solution, practitioners are often interested in knowing other similar near-optimal yet differently appearing solutions, which the search space can provide. Practitioners are often interested in this extra knowledge of multiple good solutions, so they can switich from one good solution to another in event of any changes in the market or manufacturing process. Finally, before a solution is finalized for its implementation, practitioners are often interested in exploring the *sensitivity* of the chosen solution against constraint violation. Constraints have a lower or upper bound which are usually set by participating design groups. These bounds directly affect the optimized solution, but importantly they are not usually set in stone. If a relaxation of one or more constraint bounds by a small acceptable amount can result in a significant improvement in the objective values, practitioners would like to explore this fact with the respective groups for a revision of the problem formulation. Thus, a typical industry-standard optimization task may start with a global optimization and then focus on finding multiple but diverse set of near-optimal solutions. Eventually, the task must lead into a sensitivity-based local search for providing potentially large objective-constraint trade-off solutions for practitioners to choose a final solution from. Although this work has originated from an automobile industry, the methodologies discussed here are generic and can be applied to similar other engineering design optimization tasks.

In the remainder of this paper, Section 2 presents the specific industrial problem which is represented as a black-box problem. Section 3 presents a brief survey of existing methods in solving large-scale optimization problems. Then, Section 4 presents the overall structure of the proposed many-objective methodology in three stages. Stage 1 uses an evolutionary multi-objective optimization methodology modified by a local

search approach and a number of other changes, as described in Section 5. Thereafter, Section 6 describes Stage-2 for finding multiple diverse yet equally-good solutions. Section 7 describes Stage 3 for a sensitivity analysis of a chosen solution for different levels of constraint relaxation. Sections 8, 8.1 and 9 present results of Stages 1, 2 and 3, respectively. Finally, Section 10 makes conclusions of this extensive study.

2 Optimization Problem

The goal of this paper is to solve a large-scale real-world engineering design optimization problem required by an automobile industry. To keep the confidentiality of the problem context, the industry provided the authors with an executable evaluation routine, in which a vector of 145 variables must be supplied. It then outputs a vector of 147 real numbers representing 147 performance function values and a 147×145 matrix of real numbers representing the gradient of 147 functions. Six of the first 147 performance functions are used as objective functions, and 146 of them (from second to 147-th function) are used as constraint functions. Thus, the routine can be treated as a black-box evaluation procedure for objective functions, constraint functions, and their derivatives. More detail description are provided below:

- The objective and constraint function values are known in the form of response surfaces evaluated using an executable computer program.
- The problem has 145 input variables $\boldsymbol{x} = (x_1, x_2, \cdots, x_{145})$ and 147 responses $\boldsymbol{y} = (y_1, y_2, \cdots, y_{147})$ as output.
- A single call to the evaluator for any input $\boldsymbol{x}$ returns the corresponding responses $\boldsymbol{y} = (y_1, y_2, \cdots, y_{147})$ and Jacobian $\boldsymbol{J}$. This can be summarized as

$$(\boldsymbol{y}, \boldsymbol{J}) = \texttt{eval}(\boldsymbol{x}), \tag{1}$$

  where $\dim(\boldsymbol{J}) = 147 \times 145$ and $\boldsymbol{J}_{(i,j)} = \frac{\partial y_i}{\partial x_j}$.
- Each of the 145 input variable values, i.e. $(x_1, x_2, \cdots, x_{145})$, is specified with their respective lower and upper bounds, separately, that is, $(\{l_1, u_1\}, \{l_2, u_2\}, \cdots, \{l_{145}, u_{145}\})$, and each variable is discrete taking values at an interval of 0.05 units. Units of variables and performance functions are deliberately not disclosed by the industry, except the first objective (y_1, which is the weight (in kg) of the component being designed.
- Each response $(y_2, \cdots, y_{147})$ is also specified to vary within an upper or a lower bound. This way, the problem has a total of 146×2 or 292 inequality type constraints, but at most half of them are expected to active at any solution.
- Response y_1 has no known bounds and it is one of the most important objective of the design, which is to be minimised. It is referred to as *primary* objective here.
- Additionally, the following five responses $(y_6, y_{14}, y_{29}, y_{108}, y_{146})$ must also be chosen as other conflicting objectives for the proposed optimization task. Objectives (y_6, y_{14}, y_{108}) are to be minimized, and the objectives (y_{29}, y_{146}) are to be maximized. The response y_1 is more important than the rest, meaning that although a sacrifice for other objectives to have a gain in y_1 is more desired than the other way around. Therefore, $(y_6, y_{14}, y_{108}, y_{29}, y_{146})$ objectives are called as *secondary* objectives. Due the involvement of six objectives, this problem falls into the category of many-objective optimization problem.
- Additionally, it is of great interest to the industry to seek a solution that has a get target response value of T_1 for the y_1 objective. A solution having y_1 equal to or below T_1 is desired.
- The problem is expected to have a narrow and disconnected feasible region because of 292 inequality constraints and discrete nature of variables.
- The gradient of all the responses (Jacobian) with respect to the input variables for any input $\mathbf{x}$ is also returned by the executable program and is available to be used within the optimization procedure.
- Optimized solutions need to be found with a limited computational budget of either a maximum of 50,000 calls to the evaluator or maximum of two hours of wall clock time, whichever is less.

The extent of the problem difficulty is apparent from the above description. Clearly, the overall description suggests that the underlying problem is a many-objective optimization problem. However, the problem descriptions are atypical to standard academic practices on many-objective optimization. First, a large dimensionality of the search space (145-dimensional), constraint space (146-dimensional), and objective space (six-dimensional), and discreteness of the search space are all atypical.

Second, although all six objectives must be optimized during optimization, requiring an algorithm to find at most five-dimensional trade-off Pareto-optimal solutions, one of the six objectives (representing weight of the design, y_1) is of higher importance. To handle such priorities, one may plan on using a weighted-sum approach [5,22], but designers were reluctant to provide any specific weight-vector for all the objectives, as they realized that the resulting optimized design will then intricately depend on the chosen weight-vector. One may also think of a lexicographic approach [22] in which the most important objective ((y_1) is optimized first and then a constrained optimization problem with the next important objective is formulated by ensuring that subsequent optimization tasks will not result in a degraded value of the first optimized objective. The procedure is continued until all objectives are considered and a single optimized solution is usually the outcome of the lexicographic approach. Such an approach is not suitable for this study, as designers were interested in multiple trade-off solutions.

Third, although the designers will choose a single preferred solution from the trade-off Pareto-optimal set from the optimization of six objectives with an emphasis to the weight-objective, they are curious and would like to be well-informed about other possible solutions having a similar weight but physically diverse solutions. In the optimization literature, such a task is referred to as multi-modal optimization. Population-based optimization algorithms have been redesigned to find multiple optimal solutions through the introduction of niche-preservation methods [1,10,15,21]. The multi-modal optimization methods have not been applied in the context of multi-objective optimization, where the niche formation must take place from a diversity in other five objectives, while maintaining similar values of the weight objective.

The fourth and final aspect of design requirement of this study is extremely practical and useful, and stays as one of the core contribution. After a solution has been picked from all the previous optimization considerations and is almost ready for implementation, the designers would like to make a final investigation in its neighborhood for a new solution having a potentially large trade-off between constraint relaxation and weight improvement. Constraints and their upper or lower bounds were all pre-set and an optimized solution has been found at this stage. Now, designers would like to know if they relax one or more constraint bounds slightly, to what maximum extent a weight reduction is possible. If the trade-off is significant, it is worth investigating with the corresponding domain designers about the possibility of relaxing the constraint bounds accordingly. Such a study allows a global level control of the whole design process and also helps to understand relative importance and sensitivity of the final optimized design on various constraints imposed on the problem. This comes under the scope of a sensitivity analysis often performed as a post-optimality study [25], but usually in such studies a relaxation limit must be provided to complete the analysis. What has been proposed here is a more generic approach in which a complete frontier of solutions, trading-off different limits on constraint relaxation and the corresponding maximum improvement in weight-objective are found by using a bi-objective optimization problem around the chosen solution. The idea is generic and can be applied for a single constraint, for a cluster of constraints, or for all constraints, as the case may be. These aspects of the resulting optimization tasks are summarized in the next subsection.

2.1 Targeted Optimization Tasks

The first requirement of handling a large-dimensional and discrete optimization problem is common to the other three tasks. They are handled by using a new local-search based customized optimization procedure proposed here. To make the optimization more efficient, two supplied feasible solutions are utilized in the optimization process. These two solutions are denoted as $\mathbf{w}_1^*$ and $\mathbf{w}_2^*$. The goals of the optimization procedures for handling the remaining three tasks, mentioned above, are as follows:

- **Convergence:** It is required to find significantly improved solutions from the two given solutions, specifically in achiveing $y_1(\mathbf{x}) \leq T_1$.
- **Pareto-optimality:** It is required to find a well-distributed set of trade-off solutions involving all six objectives.
- **Niche/Diversity:** It is required to find multiple Pareto-optimal solutions with similar (or almost equal) value of y_1, but each having a distinctly different variable vector.

- **Sensitivity analysis:** It is required to analyze and find other trade-off solutions in the vicinity of the optimized solution that produces a favorable trade-off between gain in one of the objective functions (y_1) and a relaxation of average constraint violation of all 292 constraints.

3 Existing Studies

Realizing that the unorthodox nature of the above-described optimization problems, it is decided to use a flexible optimization framework which is capable of handling (i) many objectives, (ii) large number of variables and constraints, (iii) discreteness of the search space, (iv) multi-modal demand of solutions, and (v) sensitivity of solutions against constraint violations. The solution approach adopted here is a population based stochastic search method which uses the principle of natural evolution and selection [14]. In this section, each problem characteristic is described and the existing literature for addressing the problems are highlighted.

3.1 Handling More Than Three Objectives

Most problems in practice are formulated as a multi-objective optimization problems involving not one, but multiple and conflicting objectives. When the number of objectives are two or three, a generic partial ordering and diversity-preserving based selection approach can be implemented in a population based EA as a reliable solution approach. NSGA-II [8], SPEA2 [32], PESA [3] are some example algorithms. However, when the number of objectives exceed three, different decomposition-based algorithms involving a guidance of parallel search directions in the objective space are needed. For many-objective (having more than three objectives) optimization problems, MOEA/D [31] and NSGA-III [11,18] are popular evolutionary algorithms. Due to the requirement of handling six conflicting objectives in this study, NSGA-III framework is used as a basic core algorithm. Studies [16] and [13] provide JAVA implementations of popular multi-objective optimization algorithms. A source code in JAVA and Matlab is also available from author's COIN laboratory web-site and also from MOEAFramework [24].

3.2 Handling a Tiny Feasible Region

With 292 constraints restricting a 145-dimensional discrete search space is likely to create fragmented and tiny feasible region. To investigate and have an idea of the density of feasible solutions in the search space, first, 2.5 million random Latin hypercube based sample solutions are created and evaluated using the black-box evaluator and are summarized in Figure 1. Interestingly, not a single feasible solution (out of 2.5 million solutions) is found. However, as shown in the inset figure of Figure 1, 17 out of 146 constraints are not violated by any of the randomly generated solutions, meaning that only 17 constraints are quite easy to satisfy, but it is extremely rare to create a feasible solution of the entire problem randomly.

The strategy to handle such a highly constrained search-space is one of the unique aspects of this study. Algorithms for handling a highly constrained search space can be found in the evolutionary optimization literature [2,17,19,23,29,30]. Among these, [17] treated feasible and infeasible solutions in separate archives and claimed to outperform the NSGA-II [8] method in a specific industrial problem. Another study [30] introduced a new parameter, called *aging* of genes, in order to avoid domination of initial population by feasible solutions early on during the optimization process, thereby maintaining a diversity of solutions in the population. Another study [29] dealt with a highly constrained combinatorial single-objective optimization problem and utilized a problem specific encoding scheme to ease the solution procedure, but the procedure is not generically applicable. A study [19] used a multi-population based parallel GA along with a repairing mechanism to reduce constraint violations in the best solution one constraint at a time. Another idea of tackling highly infeasible regions was the use of *stochastic ranking* by [26], which aimed to balance the dominance between constraint violation and objective improvement. The method aimed at reducing infeasibility in the beginning and reduced the objective value in the later stages of optimization. However, as will be described in Section 5, this study has used a systematic algorithmic approach through the use of the recently proposed multi-objectivization technique.

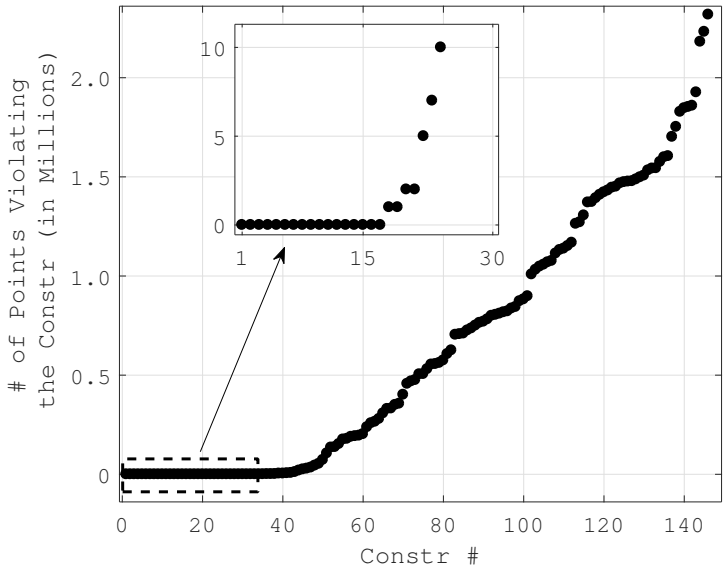


Figure 1: 2.5 million Latin hypercube samples. The x-axis is the constraint ID and the y-axis shows the number of solutions that violate them.

3.3 Handling a Limited Computational Budget

One of the key requirements to this application study is to find solutions within a computational budget of a maximum of 50,000 calls to the evaluator, or a maximum wall clock time of 2 hours, whichever happens earlier. In many other engineering optimization studies, 50,000 function calls are already quite a large computational effort, but due to the use of response surfaces in the optimization process requiring about 0.1 to 0.2 seconds to evaluate each solution, the industry generously allowed a maximum of 50,000 calls to the evaluator for the entire optimization task.

A large volume of studies exist for the use of a surrogate modeling approach in an optimization procedure to handle a low budget of function calls. The study [27] provides a state-of-the-art account of applying surrogates in engineering optimization. Since this supplied evaluation procedure is already constructed with surrogate models obtained from high-fidelity solutions evaluations, there is no additional benefit of using another surrogate modeling effort here, but a progressive surrogate modeling approach may provide a better outcome, if the industry was allowed to integrate surrogate modeling and optimization. Hence, an extensive literature on the topic is not discussed here. An evolutionary optimization method with an archiving strategy such as archive-based micro GA (AMGA) [28] is an excellent way to keep the computations low by keeping the population size small.

A budget of 50,000 solutions may seem large at the first instance, but for a 145-dimensional search space it is not adequate for a standard off-the-shelf optimization algorithm to get close to the optimal region. As a result, without a local search strategy focusing in potential areas of the search space, achieving a fast convergence will be challenging, even if some known good solutions $(\mathbf{w}_1^*, \mathbf{w}_2^*)$ were provided. Fortunately, gradients of the response surfaces for objectives and constraints are available as a bonus for this problem. Therefore, unlike most other evolutionary optimization applications, the proposed framework exclusively utilizes the gradient information in an integrated manner with standard evolutionary operators to constitute an efficient search process, thereby making this study unique in terms of developing a hybrid point-and-population based optimization algorithm.

3.4 Handling Discrete Variables

Discrete nature of all 145 variables present another challenge for search algorithms to work well and to constitute a gradual search, as the discreteness in variables break the continuity of the response surfaces and hence make their gradients somewhat meaningless. A past study [20] showed a sound neighborhood search concept for categorical variables in an engineering design problem, requiring gradients of the objectives.

It is important to mention that due to the flexibilities of representing a solution with an evolutionary optimization framework, these methods usually perform well on discrete search spaces. A recent study [12] using a population based algorithm has solved an integer linear programming problem arising from an industrial resource allocation problem involving more than billion variables in a polynomial computational time complexity.

The above discussions reveal that population-based evolutionary optimization algorithms are flexible and can be modified to solve different kinds of optimization problem solving tasks of this study. The next section proposes the methodology developed here.

4 Proposed Methodology: MOTRAN

The optimization methodology must minimize four objectives: y_1, y_6, y_{14} and y_{108} and simultaneously maximize two other objectives: y_{29} and y_{146}. The latter two objectives are multiplied by -1 and all six resulting objectives are then minimized. The proposed methodology is called as *Multi-objective Trade-off Analyzer* or *MOTRAN*, in short. The work flow of MOTRAN is composed in three stages:

- **Stage-1:** As shown in the top part of Figure 2, this stage aims to find a set of diverse feasible non-dominated solutions ($\mathcal{F}$) that are within an acceptable range in the response y_1, but the solutions make a trade-off of all six objectives mentioned above. This stage utilizes NSGA-III coupled with a customized local-search method for a faster convergence near the six-objective Pareto-optimal set.

- **Stage-2:** In this task, a pair of trade-off solutions are identified from the obtained non-dominated set $\mathcal{F}$ which have almost equal y_1 values, but very different values of other five objectives. To achieve this task, care has been placed in Stage-1 optimization to obtain requisite diverse solutions for each high-yield y_1 solution. This will provide designers with two or more different high-performing designs, but having similar y_1 value (weight, in this case).

- **Stage-3:** As shown in the bottom part of Figure 2, this stage explores new solutions in the vicinity of a chosen non-dominated solution from $\mathcal{F}$, all of which make an optimal trade-off between constraint relaxation and improvement in weight objective y_1.

A description of each of these stages is given below.

Total allocation of 50,000 evaluations are distributed as follows. Stage-1 consumes 30,000 solution evaluations (SEs) in approximately 1 hr of computational time. Stage-2 does not consume any new solution evaluation, as it searches for pairs of similar y_1 solutions from $\mathcal{F}$. The computational time is also negligible compared to that in Stage-1. Stage-3 takes another 5,000 to 10,000 SEs, consuming about 0.25-0.5 hrs, totaling 1.5 hrs on a modest desktop computer. Since the black-box function is provided as a static object (that is, `.so` file), the individual function calls may be slow. If the black-box is completely integrated into MOTRAN by providing the response surfaces as mathematical objects, surely the computational time will be shorter. For this reason, results are shown here in terms of SEs and not in terms of the wall clock time.

5 Stage-1: Many-objective Search Aided by Gradient-based Local Search

This stage conducts the main multi-objective search to attempt to find a set of Pareto-optimal solutions for six objectives, which are

- diverse in the 145-dimensional design space

- diverse in the six-dimensional objective space, and,

- in an acceptable range of y_1 values with $T_1 = 174$: $y_1(\mathbf{x}) \leq 174$, specified by the industry.

A six-objective optimization problem is solved in this stage using a variant of NSGA-III algorithm [11, 18], aided by a newly proposed gradient-based local search method.

In the presence of constraints in an optimization problem, the optimum solution usually lies on the intersection of boundaries of critical constraints. Moreover, since the problem has 146 constraints, there is a large possibility that the search space contains multiple feasible 'islands'. Therefore, a population

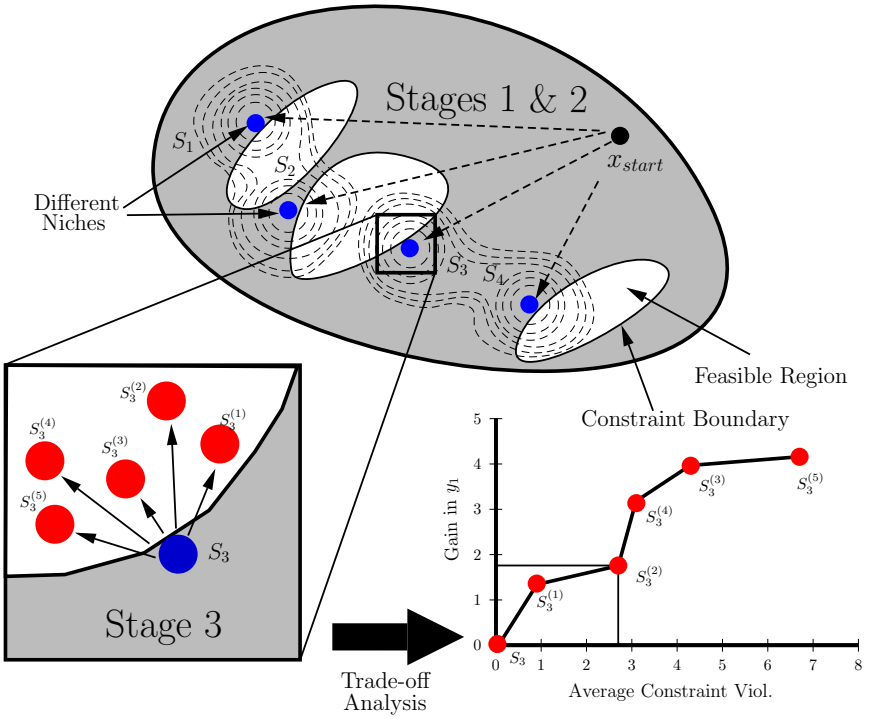


Figure 2: The proposed three-stage procedure of MOTRAN. In Stage-1, it focuses on finding multiple trade-off and non-dominated solutions for six conflicting objectives. Stage-2 identifies pairs of trade-off solutions from Stage-1 which have similar values of y_1, but different values of other objectives. Stage-3 makes a sensitivity analysis of a chosen solution by means of a bi-objective optimization in its neighborhood in identifying a trade-off between constraint relaxation and gain in objective y_1 value.

based search method, such as NSGA-III, must attempt to explore parallely as many such feasible islands as possible in order to locate the most promising regions quickly. However, such 'exploration-centric' mechanism must be aided with an 'exploitation' phase to allow quicker convergence properties of the overall algorithm. For this purpose, an efficient local search (LS) algorithm, presented in Algorithm 1, is devised based on gradient information provided by the evaluator. The proposed LS algorithm uses a *feasible steepest descent search*. Despite the use of gradients, an intriguing aspect of the LS method is that it should work on the discrete variable space. A detailed explanation of the LS algorithm is discussed in the next subsection.

5.1 Local Search: Feasible Steepest Descent Search

Algorithm 1 shows the local search (LS) algorithm that Stage-I of MOTRAN uses to introduce a faster convergence behavior of NSGA-III. NSGA-III uses an archive ($\mathbf{\Pi}$) of feasible solutions found since the beginning of the optimization run. Whenever a new feasible solution is added to $\mathbf{\Pi}$, NSGA-III calls the LS procedure to change the solution to a locally better solution in its neighborhood. The search direction for the LS comes from a compromise of feasibility and descent properties of y_1, similar to an idea proposed by Zoutendijk [33]. Before Algorithm 1 is described, some details about the problem are provided below:

- The problem has $n = 145$ variables $\{x_1, x_2, \ldots, x_n\}$ and $n_r = 147$ response surfaces, say $\{y_1, y_2, \ldots, y_{n_r}\}$.
- No constraint is formed with y_1, but all other $n_c = n_r - 1$ or 146 response surfaces are used to form constraints.
- The list $\boldsymbol{\tau} = \{\tau_1, \tau_2, \ldots, \tau_{n_c}\}$ carries information on the 'type' of each constraint. For example, i-th constraint g_i formed based on response surface y_{i+1} may be of type $\tau_i =$ `UPPER_BOUND` or `LOWER_BOUND`.
- The list $\boldsymbol{\beta} = \{\beta_1, \beta_2, \ldots, \beta_{n_c}\}$ carries information on the value of bound on a response value for the corresponding constraint. For example, if constraint g_i based on response y_{i+1} has $\tau_i =$ `UPPER_BOUND`, then constraint g_i has bound value of β_i and $g_i \equiv y_{i+1} \leq \beta_i$.

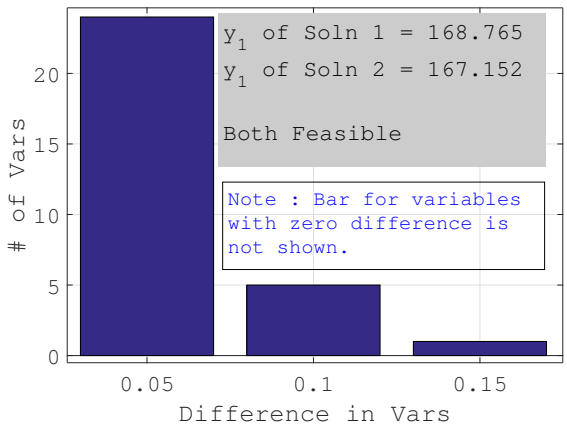


Figure 3: The bar graph shows the difference between the $y_1 = 168.765$ solution and the solution obtained after applying the local search to reach the $y_1 = 168.765$ solution.

- The lists $\boldsymbol{l} = \{l_1, l_2, \ldots, l_{n_v}\}$ and $\boldsymbol{u} = \{u_1, u_2, \ldots, u_{n_v}\}$ contain the lower and upper bounds of the variables, that is, $l_i \leq x_i \leq u_i$.

The above information are provided by the industry.

Algorithm 1 is now described in detail.

- *Input:* The solution to be locally searched is provided as an input: $\boldsymbol{w} = \{w_1, w_2, \ldots, w_n\}$. The lists τ , β, $\boldsymbol{l}$ and $\boldsymbol{u}$ are also provided. A small tolerance `tol` is used to determine the satisfaction of a constraint is also an input. A parameter `tol` is provided as an input. It is used to relax the lower or upper bound, as the case may be, of each constraint to establish whether a solution is feasible.

- *Output:* Algorithm 1 returns the final feasible locally searched solution $\boldsymbol{v}$ and the updated archive of feasible solutions $\boldsymbol{\Pi}$. Algorithm 1 not only adds the final improved, feasible, and locally searched solution $\boldsymbol{v}$ to the archive $\boldsymbol{\Pi}$, but also adds all the intermediate feasible solutions having a lower y_1 value than that of the original solution $\boldsymbol{w}$.

- *Lines 1-6 :* Line-1 initializes a number of basic variables for the algorithm. Line-2 initializes the solution $\boldsymbol{x}$ with $\boldsymbol{w}$. Variable H keeps a count of number of calls to the program `eval()` given in Equation 1. Variable f is a flag which is set to 1, if no feasible descent solution is possible in a single discrete step of $\pm$ 0.05. Line-3 starts a 'while' loop to search for a solution which is feasible and has lowest response value for y_1 that could be found. Line-4 updates the variable $\boldsymbol{v}$ with the current best-known solution $\boldsymbol{x}$. Lines 5 and 6 are self-explanatory.

- *Lines 7-8* Line-7 calls the procedure `CV()` given in Algorithm 2 to calculate the list of normalized constraint violations $\boldsymbol{g} = \{g_1, g_2, \ldots, g_{n_c}\}$ and the transformed Jacobian matrix $\boldsymbol{G}$ for a tolerance value of *zero*. The transformation is done in a way such that $\boldsymbol{G}_{(1,q)} = {}^{\partial y_1}/_{\partial x_q}$ and $\boldsymbol{G}_{p,q} = {}^{\partial g_p}/_{x_q}$, where $p \in \{2, 3, \ldots, n_r\}$ and $q \in \{1, 2, \ldots, n_v\}$. Here g_i refers to the normalized constraint violation value within some predefined tolerance corresponding to response y_{i+1}. Constraints are modified so that $g_i(\mathbf{x}) > 0$ means that the solution $\mathbf{x}$ does not satisfy the constraint. Line-8 increments i-th variable by δ_i in steps of ± 0.05.

- *Lines 9-16* In the for-loop, the jump in a variable is decided depending upon the values of partial derivative $G_{(1,i)} = {}^{\partial y_1}/_{\partial x_i}$ and the bounds of the variable x_i. The parameter δ_i in x_i is set to 0.05 or -0.05, if it has the possibility of reducing the value of y_1 (that is, if $G_{1,i} < 0$, or $G_{1,i} > 0$, respectively) without violating the variable bounds $l_i \leq x_i \leq u_i$. Otherwise, δ_i is set to zero.

- *Lines 17-18* Line-17 checks if no jumps are possible for $\boldsymbol{x}$ to reduce y_1 without violating the corresponding variable bounds and consequently breaks the *while* loop of Line-3 by setting flag f to one. Otherwise, the algorithm continues.

Algorithm 1 LS(): A procedure to conduct a local search on a given feasible solution.

input: $\boldsymbol{w}$, $\boldsymbol{\tau}$, $\boldsymbol{\beta}$, $\boldsymbol{l}$, $\boldsymbol{u}$, $\texttt{tol}$, $\boldsymbol{\Pi}$ ▷ $\boldsymbol{w}$ is feasible.
output: $\boldsymbol{v}$, $\boldsymbol{\Pi}$

$n_v \leftarrow 145,\ n_c \leftarrow 146,\ n_r \leftarrow 147$ ▷ n_v : # Variables, n_c : # Constraints, n_r : # Responses
$\boldsymbol{x} \leftarrow \boldsymbol{w},\ n \leftarrow 0,\ f \leftarrow 0$ ▷ $\dim(\boldsymbol{w}) = 1 \times n_v$
while $f \neq 1$ **do**
  $\boldsymbol{v} \leftarrow \boldsymbol{x}$
  $(\boldsymbol{y}, \boldsymbol{J}) \leftarrow \texttt{eval}(\boldsymbol{x})$ ▷ $\dim(\boldsymbol{y}) = 1 \times n_r,\ \dim(\boldsymbol{J}) = n_r \times n_v,\ J_{(p,q)} = \partial y_p / \partial x_q$
  $n \leftarrow n + 1$
  $(\boldsymbol{g}, \boldsymbol{G}) \leftarrow \texttt{cv}(\boldsymbol{y}, \boldsymbol{J}, \boldsymbol{\tau}, \boldsymbol{\beta}, 0)$ ▷ $\dim(\boldsymbol{g}) = 1 \times n_c, \dim(\boldsymbol{G}) = n_r \times n_v$
  $\boldsymbol{\delta} \leftarrow \boldsymbol{0}$ ▷ $\dim(\boldsymbol{\delta}) = 1 \times n_v$
  for $i \in \{1, 2, \ldots, n_v\}$ **do**
    if $(G_{(1,i)} > 0) \wedge (x_i - 0.05 \not< l_i)$ **then**
      $\delta_i \leftarrow$ -0.05 ▷ Only steps of ± 0.05 allowed.
    end if
    if $(G_{(1,i)} \leq 0) \wedge (x_i + 0.05 \not> u_i)$ **then**
      $\delta_i \leftarrow 0.05$
    end if
  end for
  if $(\boldsymbol{\delta} = \boldsymbol{0})$ **then**
    $f \leftarrow 1$
  else
    $\boldsymbol{\Delta} \leftarrow \big[\ \underbrace{\boldsymbol{\delta}^\top \quad \boldsymbol{\delta}^\top \quad \cdots \quad \boldsymbol{\delta}^\top}_{n_r \text{ times}}\ \big]^\top$ ▷ $\dim(\boldsymbol{\Delta}) = n_r \times n_v,\ \boldsymbol{\Delta}_{(p,q)} = \delta_q$
    $\boldsymbol{\Theta} \leftarrow \boldsymbol{G} \circ \boldsymbol{\Delta}$ ▷ $\circ$ is Hadamard product.
    $\boldsymbol{\xi} \leftarrow \boldsymbol{0}$ ▷ $\dim(\boldsymbol{\xi}) = 1 \times n_c$
    for $i \in \{1, 2, \cdots, n_v\}$ **do**
      $\xi_i \leftarrow \sum_{j=1}^{n_c} (g_j + \Theta_{(j+1,\, i)} - \texttt{tol})$
    end for
    $i^* \leftarrow -1,\ \theta^* \leftarrow 0$
    for $i \in \{1, 2, \cdots, n_v\}$ **do**
      if $(\underbrace{\xi_i \leq 0}_{Feasible}) \wedge (\underbrace{\Theta_{(1,i)} < \theta^*}_{\text{Descent in } y_1})$ **then**
        $i^* \leftarrow i,\ \theta^* \leftarrow \Theta_{(1,i)}$
      end if
    end for
    if $(\theta^* = 0)$ **then**
      $f \leftarrow 1$
    else
      $\hat{\boldsymbol{x}} \leftarrow \boldsymbol{x}$
      $\hat{x}_{i^*} \leftarrow \hat{x}_{i^*} + \delta_{i^*}$
      $(\hat{\boldsymbol{y}}, \hat{\boldsymbol{J}}) \leftarrow \texttt{eval}(\hat{\boldsymbol{x}})$
      $n \leftarrow n + 1$
      $(\hat{\boldsymbol{g}}, \hat{\boldsymbol{G}}) \leftarrow \texttt{cv}(\hat{\boldsymbol{y}}, \hat{\boldsymbol{J}}, \boldsymbol{\tau}, \boldsymbol{\beta}, \texttt{tol})$
      if $(\hat{y}_1 < y_1) \wedge (\sum_{i=1}^{n_c} \hat{g}_i = 0)$ **then**
        $\boldsymbol{x} \leftarrow \hat{\boldsymbol{x}}$
        $\boldsymbol{\Pi} \leftarrow \boldsymbol{\Pi} \cup \boldsymbol{x}$ ▷ Add to archive of NSGA-III.
      else
        $f \leftarrow 1$
      end if
    end if
  end if
end while
return $(\boldsymbol{v}, \boldsymbol{\Pi})$

Algorithm 2 CV(): A procedure to calculate constraint violation and the normalized Jacobian from the responses of a solution.

input: $\boldsymbol{y}, \boldsymbol{J}, \boldsymbol{\tau}, \boldsymbol{\beta}, \texttt{tol}$
output: $\boldsymbol{g}, \boldsymbol{G}$

$n_v \leftarrow 145,\ n_c \leftarrow 146,\ n_r \leftarrow 147$ ▷ n_v : # Variables, n_c : # Constraints, n_r : # Responses
$\boldsymbol{g} \leftarrow \boldsymbol{0}, \boldsymbol{G} \leftarrow \boldsymbol{J}$ ▷ $\dim(\boldsymbol{g}) = \dim(\boldsymbol{g}) = 1 \times n_c, \dim(\boldsymbol{G}) = n_r \times n_v$
for $i \in \{1, 2, \ldots, n_c\}$ **do**
    if $\tau_i ==$ `UPPER_BOUND` **then** ▷ Constraint has an upper bound.
        $m \leftarrow 1$
    else ▷ Constraint has a lower bound.
        $m \leftarrow -1$
    end if
    if $|\beta_i| > 1$ **then**
        $g_i \leftarrow \left\{\left(\frac{y_{i+1}}{\beta_i}\right) - 1\right\} \cdot m$
        for $j \in \{1, 2, \ldots, n_v\}$ **do**
            $G_{(i+1,\ j)} \leftarrow \left\{(1/\beta_i) \cdot J_{(i+1,\ j)}\right\} \cdot m$
        end for
    else
        $g_i \leftarrow \{y_{i+1} - \beta_i\} \cdot m$
        for $j \in \{1, 2, \ldots, n_v\}$ **do**
            $G_{(i+1,\ j)} \leftarrow \left\{J_{(i+1,\ j)}\right\} \cdot m$
        end for
    end if
    $g_i \leftarrow g_i - \texttt{tol}$
    if $g_i < 0$ **then** ▷ Feasible case.
        $g_i \leftarrow 0$
    end if
end for
return $(\boldsymbol{g}, \boldsymbol{G})$

- *Lines 19-21* A jump matrix $\boldsymbol{\Delta}$ is created from $\boldsymbol{\delta}$ such that $\boldsymbol{\Delta}_{(p,q)} = \delta_q$ where $p \in \{1, 2, \ldots, n_r\}$ and $q \in \{1, 2, \ldots, n\}$. Line-21 calculates the *Hadamard product* of transformed Jacobian $\boldsymbol{G}$ and jump matrix $\boldsymbol{\Delta}$ and stores it in the matrix $\boldsymbol{\Theta}$. $\Theta_{(1,q)}$ refers to the possible change in y_1 because of δ_q change in x_q of $\boldsymbol{x}$. Similarly, $\Theta_{(p,q)}$ refers to the possible change in normalized constraint g_{p-1} because of jump δ_q in variable x_q of $\boldsymbol{x}$. Here, $p \in \{2, 3, \ldots, n_r\}$ and $q \in \{1, 2, \ldots, n\}$.
- *Lines 22-31* Line-22 initializes the list $\boldsymbol{\xi} = \{\xi_1, \xi_2, \ldots, \xi_{n_v}\}$ with zeros. In lines 23-25, the variable ξ_i stores the possible value of total normalized constraint violation over all constraints due to a jump of δ_i in variable x_i of $\boldsymbol{x}$. Lines 26-31 find the index i^* of variable jump δ_i in x_i that corresponds to having the maximum potential of reducing response value y_1 by an amount of θ^*, while still being feasible.
- *Lines 32-33* Lines 32-33 terminate the procedure, if there is no i^* in $\{1, 2, \ldots, n_v\}$ with a corresponding $\theta^* < 0$. If a jump with a potential to reduce y_1 while maintaining feasibility is found, then the algorithm continues.
- *Lines 34-end* Line 35-36 create a new variable $\hat{\boldsymbol{x}}$ from $\boldsymbol{x}$ by adding a jump of δ_i to variable x_i of $\boldsymbol{x}$. Line 37-38 call the evaluator to make an actual evaluation on $\hat{\boldsymbol{x}}$ to obtain corresponding responses $\hat{\boldsymbol{y}}$ and increases the total function evaluations H by one. Line-39 calls the procedure `CV()` (Algorithm 2) to obtain corresponding normalized constraint violation values $\hat{\boldsymbol{g}}$ within the desired tolerance `tol`. Line-40 checks if the response $\hat{y}_1$ of solution $\hat{\boldsymbol{x}}$ is indeed smaller than y_1 of solution $\boldsymbol{x}$ and whether the solution $\hat{\boldsymbol{x}}$ has a zero total constraint violation. If both conditions are found to be true, then solution $\boldsymbol{x}$ is updated in Line-41 and subsequently added to the archive $\boldsymbol{\Pi}$. If the aforementioned conditions are not met, then flag variable f is set to one to break the *while* loop of line-3. Subsequently, the algorithm returns the best solution found, $\boldsymbol{v}$, and the updated archive $\boldsymbol{\Pi}$ to the calling program.

Algorithm 1 is applied to each feasible solution discovered by NSGA-III during the MOTRAN Stage-1. Furthermore, the starting solution, the final solution, and all intermediate feasible solutions of a local search are archived so that any future local search from these points is carried out only once in an NSGA-III run. A new local search operation is initiated only if any new feasible solution is not present in the archive.

5.2 Modified NSGA-III Procedure

The original NSGA-III procedure [11] is described next. The algorithm starts with a parent population of randomly created solutions. The size of the population can be set as equal to the number of desired Pareto-optimal solutions. To begin with, a set of well-distributed set of reference directions [4] originating from the origin are pre-specified in the first coordinate system of the m-dimensional objective space (m is the number of objectives). One generation of NSGA-III procedure works as follows. An offspring population of the same size as the parent population size is created by using algorithm's operators, which can be customized using problem information to have a more efficient application. The populations are combined to form a merged population. First, all objective values of the merged population members are normalized so that the resulting non-dominated solutions lie within [0,1]. This allows the population members to be compared with pre-specified reference lines. Each population member is associated with its nearest reference line and in turn, all associated members of each reference line are identified and grouped together. The merged population is then ranked according to increasing levels of constraint non-domination [5, 8] using all m objectives and constraints. Since level 1 solutions are infinitely better than subsequent levels, all solutions are accepted level-wise starting from level 1, until no more levels can be accepted to fill the new population slots. Note that only half of the merged population can be accepted to make the overall process a systematic algorithm. The last level of solutions which could not be accepted as a whole to meet the population size requirement are then considered for the final niching operator to select a well-diversed set of remaining population members. For each reference line, its nearest associated member is chosen for this purpose. It is important to mention that in large-objective problems, almost all merged population members lie on level 1 of non-domination after a few generations, and the final niching operator becomes the main selection operator of the NSGA-III procedure. Due to the emphasis of non-dominated solutions and well-diversified solutions through individual picking of closely adhered solutions of a set of well-distributed reference lines, NSGA-III is able to converge with a good distribution of non-dominated and trade-off solutions at the end of a simulation run.

For this study, the above-described NSGA-III procedure needs to be modified to have the following updates:

1. Initial population is seeded with two supplied feasible solutions.
2. Ensure the survival of the best y_1 solution ($\mathbf{x}^{(1*)}$) at every generation.
3. Ensure survival of all distant variable-space solutions of $\mathbf{x}^{(1*)}$. This is enforced to facilitate Stage-2 task, as no additional optimization is performed in Stage-2.

All the updates performed on the original NSGA-III procedure are described below.

5.2.1 Initial Population Update and Archive Creation

The supplied feasible solutions are included in the initial population and the rest of the solutions are created at random. Initial reference directions are specified uniformly over the entire positive objective space, as in the original NSGA-III.

To preserve good y_1 solutions found by NSGA-III procedure, an archive is used to store all feasible solutions discovered thus far. To start with, the archive contains the two supplied feasible solutions and any other feasible solutions that may have been created at random.

5.2.2 Merged Population Update

At every generation, the merged population is created by using, not only the parent and offspring populations, but also including the current archive. Duplicates, if any, in the merged population – based on objective space similarity in response y_1 within a small tolerance of say ϵ_y[1]– are removed from the merged population. The usual normalization and association operations are then performed using the existing reference directions. The merged population is sorted according to constraint non-domination principle and level-wise solutions starting from the first level are accepted. The niche-preservation operator applied to the last accepted level of solutions is updated, as described in Section 5.2.4, but before that a simple and clever strategy is proposed for ensuring survival of a desired solution from one generation to the next.

5.2.3 Ensuring Survival of a Desired Solution

NSGA-III uses a set of well-distributed reference directions in the objective space, and tries to find a single Pareto-optimal solution along each reference direction. This is why the population size of NSGA-III is set as the number of reference directions. In NSGA-III's association operator, a solution is assigned to be associated with a reference line, if it has the shortest perpendicular distance from the line than all other lines. Thus, in order to ensure that the a desired solution is preserved between two generations, one way to achieve this is to allocate a new reference direction joining origin to this solution in the objective space. This will make the solution to have a perpendicular distance zero from this newly assigned reference line and will be declared as the associated member of the new reference line.

5.2.4 Niche-Preserving Selection Operator Update

First, the feasible solutions of the last accepted non-dominated level are identified. A new normalization of these feasible solutions is performed based on their extreme points. These normalized feasible solutions are compared against the existing reference points and association of solutions with reference directions are made. Note that there can some un-important reference directions without any associated feasible population member. Next, these solutions are sorted in ascending order of y_1 objective value. Thereafter, the best y_1 solution (top of the sorted list) is chosen and all other solutions which lie within a user-supplied ϵ_x parameter in the variable space are *cleared* from the sorted list. Then, the next solution in the list is chosen and clearing operation is performed again and this process is continued all feasible solutions of the last level of domination is executed. This process will end up in a finite set of picked solutions, which are (i) good in terms of y_1 objective, and (ii) well diversed from each other in the variable space. As many of these good set of feasible solutions (elites) as possible are then attempted to be preserved and saved in the new population for the next generation.

Based on the number of elite solutions (n_e), the available population slots (n_{rem}), and number of un-important reference directions (n_b), there can be different scenarios. First, un-important reference directions are replaced with new reference directions created from elite solutions with a procedure described

[1]An ϵ_y of 10^{-3} is used in this study.

in the previous subsection. If $n_e < n_b$, then randomly chosen n_e un-important reference directions are replaced with elite-based directions; else, all un-important reference directions are replaced. In the case of latter, or when $n_b = 0$ (meaning that no reference direction is empty), some of the existing reference directions are replaced with remaining elite-based reference directions. In the event of n_e being more than total number of reference directions, elites are chosen randomly and all existing reference directions are replaced by the chosen elite-based directions. After the reference directions are updated by emphasizing elite and diverse y_1 solutions from the merged population, association of solutions with updated reference directions are made again and the usual NSGA-III niching operation is then performed.

5.2.5 Local Search Before Start of New Generation

Every feasible solution of the newly formed population is checked for their local search update. If a solution is not local-searched before, a local search is performed based on the procedure described before and the intermediate and final feasible solutions are stored in the archive.

All these update procedures ensure that any new feasible and diverse solution is searched locally to discover any new better feasible solution in its vicinity in the variable space and once such a solution is found, the updates ensure that they are preserved until a better solution is discovered.

6 Stage-2: Identifying Pair-wise Trade-off Solutions Having Similar y_1

Once the stage-1 optimization algorithm ends, the next step is to find a set of pair-wise solutions that belong to different niches, but have the same or similar y_1 values. For this purpose, two user-defined parameters are used:

- Δy_1: The maximum difference in y_1 within which two solutions can be considered close to each other, and,
- Δx: The minimum variable space distance, which is expected to exist between the two solutions for them to be different from each other.

Although in this study, only a pair of such similar yet well-niched solutions are attempted to be identified, more than two such solutions can also be found by the proposed procedure, described next. The search for a pair is restricted only to non-dominated set obtained after Stage-1 run. First, all solutions are ordered from smallest y_1 to largest y_1 values. Then, for each solution $\mathbf{x}^{(i)}$, the set of solutions within a distance of Δy_1 from $\mathbf{x}^{(i)}$ are checked further. Then, all solutions with a minimum distance of Δx from $\mathbf{x}^{(i)}$ are identified and the one with the largest Δx from $\mathbf{x}^{(i)}$ is chosen as pair. The above process can be continued until all such pairs are identified or a pre-specified number of pairs are found. A simple change in the above procedure can be made to select more than two similar solutions, if desired.

7 Stage-3: Sensitivity Between Constraint Relaxation and Objective Improvement

Stage-3 starts with a chosen non-dominated solution from Stage-1 and Stage-2 tasks. Say, the chosen solution is $\mathbf{x}^*$. To investigate the effect of constraint relaxation on the objective (y_1) improvement, a bi-objective optimization problem is formulated at $\mathbf{x}^*$. Objectives in conflict are as follows: (i) minimization of total normalized constraint violation [9], and (ii) maximization of gain in objective y_1 reduction from $y_1(\mathbf{x}^*)$. The resulting Pareto-optimal solutions from this bi-objective optimization will reveal the maximum gain in y_1-objective possible for a given relaxation of constraint violation.

NSGA-II [8] is used for this purpose with the following change. The initial population is created by mutating each of the 145 variables of the chosen solution $\mathbf{x}^*$ by a random value between $-v_{step}$ to $+v_{step}$ in discrete steps of 0.05. The value v_{step} is chosen by the user. Lower values of v_{step} will limit the search to very close by region around $\mathbf{x}^*$, whereas higher values of v_{step} will be too much of search space to manage with the relatively small computational budget of Stage-3. After experiments, a v_{step} value of 2-5 is chosen.

Table 1: Parameter settings for Stage-1.

Parameter	Description	Suggested Value
pop	Population Size	84
maxfe	Max Func. Evals	30,000
p_c	Probability of Crossover	0.9
p_m	Probability of Mutation	15/145
η_c	Distribution index for SBX [7]	20
η_m	Distr. index for Polynomial Mutation [8]	20
ctol	Constraint Tolerance (in %)	0.1 – 0.5

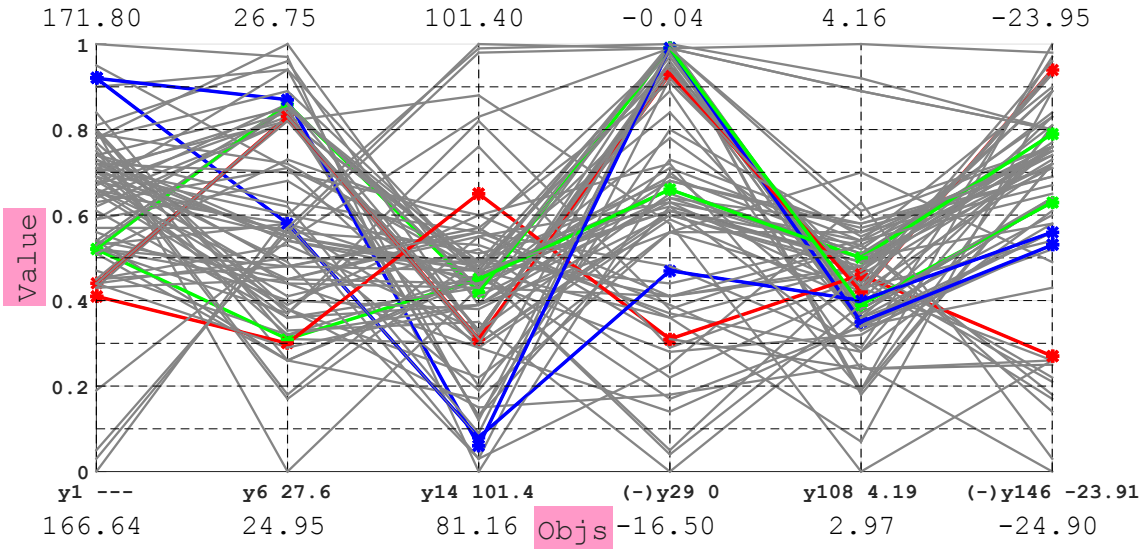


Figure 4: Trade-off solutions obtained at the end of Stage-1 of MOTRAN.

8 Results from Stage-1

In this section, Stage-1 results are presented first. Table 1 shows the parameter values for the Stage-1 optimization. Best known solutions with $y_1(\mathbf{w}_1^*) = 184.20$ and $y_1(\mathbf{w}_2^*) = 168.76$ are introduced in the starting population. Stage-1 optimizer with the parameter values summarized below consistently yielded a set of solutions from different niches, as shown in Figure 4. It is interesting to note that Stage-1 procedure has found a better solution with $y_1 = 166.64$ than the best supplied solution having $y_1 = 168.76$. Moreover, all 84 trade-off solutions lie below $y_1 = 171.800$.

The effectiveness of the local search operator is presented next. Figure 5 shows the value of y_1 before and after the local search procedure of all feasible solutions which were local searched during the optimization run. The fact that almost all y_1 values after local search are smaller than that before local search, the local search is able to improve a solution in most cases. The distance of a point from the 45-degree solid line signifies the amount of improvement in y_1 value. It is clear that as the solutions get closer to the optimum, the extent of improvement by the local searth gets smaller. This also means that the amount of improvement is not large, meaning that the proposed local search cannot be used to achieve a global search. The reduction in y_1 values come from NSGA-III's generations. The consideration of six objectives during the optimization task allows a diversity in solutions from start to end of the proposed algorithm, as shown by three sets of solutions during start to 10,000 FEs, between 10,000 to 20,000 FEs, and the remaining FEs. The figure shows also that a y_1 of 166.74, something very close to the best y_1 value of 166.64 found in Stage-1, does not come from the best y_1 value of 168.76 supplied in initial population before local search, rather it comes from a solution with $y_1 = 169.03$ and during the intermediate part of the optimization run (between 10,000 to 20,000 FEs).

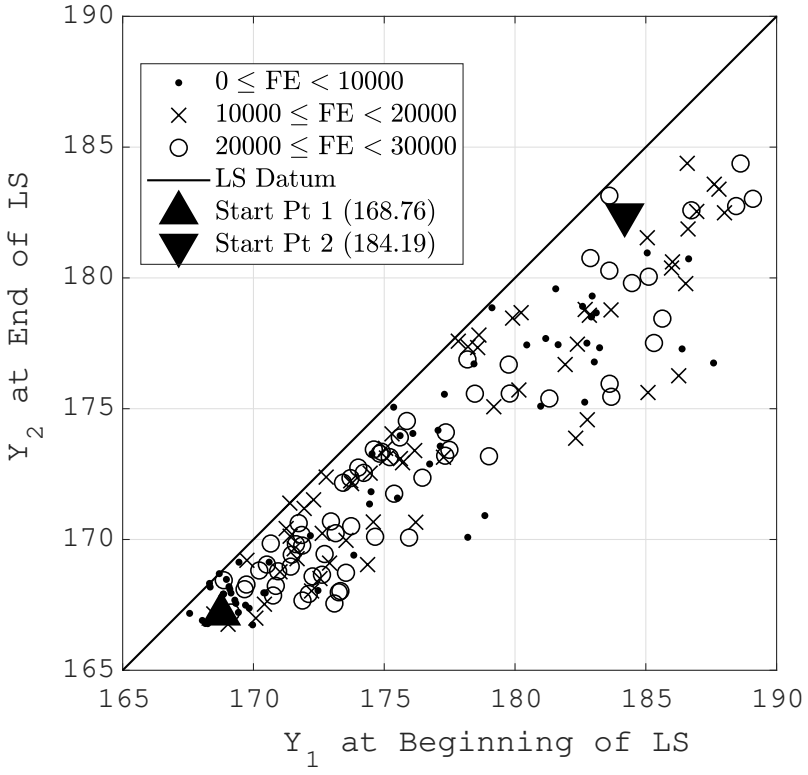


Figure 5: A scatter plot showing the y_1 value at the beginning and end of the local search procedure for different solutions during Stage-1 of MOTRAN.

8.1 Results from Stage-2

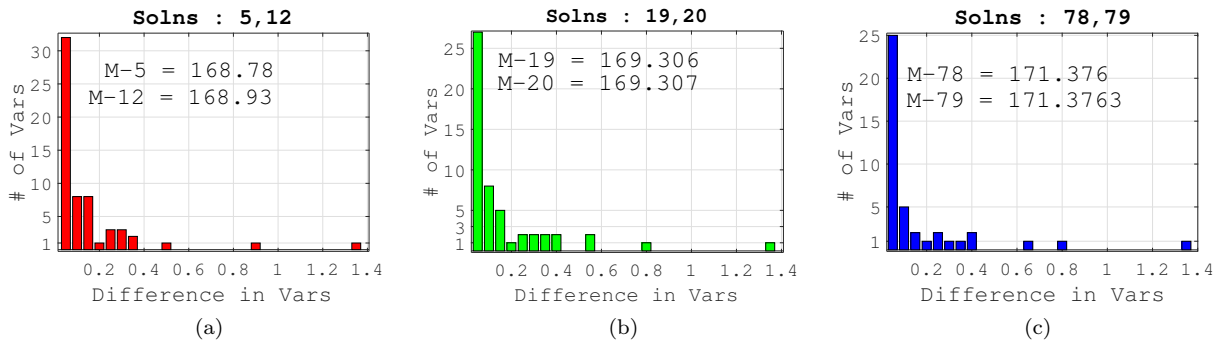
Stage-2 approach is then applied to Stage-1 solutions to obtain three pairs of solutions, for which each pair has similar y_1 values but has a large difference in the variable space. Figure 4 shows that each pair has starts almost at the same point on the first vertical axis marking y_1 objective, but differs in other objectives. Figure 6 shows the statistics of three pairs of solutions. The x-axis marks the amount of difference in variable values between two solutions in a pair and the y-axis marks the number of such variables. For example, the first pair has 32 variables with a difference of 0.05 (one step), eight variables with a difference of 0.1 (two steps), eight variables with a difference of 0.15 (three steps), and so on. Despite being difference, the y_1 values have a difference of only 0.09% from each other. Similar differences are observed in the other two pairs.

It is interesting to note that no such pairs are found for the minimum y_1 solution ($y_1 = 166.64$), indicating that the optimal region of the y_1 objective is comes from similar solutions for this problem. To find two different solutions providing a similar y_1 occurs only at $y_1 = 168.78$, about 2.14 units of y_1 away from the optimized solution. These are interesting and valuable information for a designer to make a decision in choosing between a lone well-optimized solution versus a variety of near-optimized but differently appearing solutions.

9 Results from Stage-3

The best feasible solution $\mathbf{x}^*$ found from the Stage-1 has a y_1 value of 166.64. Stage-3 procedure is now applied to this solution. Table 2 shows the parameter settings used for the Stage-3 optimizer. A constraint tolerance of 100% is used here, as the constraint violation is used as an objective.

Figure 7 shows the trade-off solutions obtained by NSGA-II procedure. The origin of this plot indicates the solution $\mathbf{x}^*$ obtained from Stage-1 optimizer. It has no constraint violation and no difference in y_1 value from itself. The point marked vertically up from origin indicates that without any constraint violation, an improvement of about 0.08 kg of y_1 is achievable, meaning that modified NSGA-III in Stage-1 with all its handling of six objectives missed finding this solution which is better than $\mathbf{x}^*$. Due to focus of search by



(a) (b) (c)

Figure 6: Histograms of variable values of three pairs of similar y_1 solutions from Stage-1 trade-off solutions.

Table 2: Parameter settings for Stage-3 optimization.

Parameter	Description	Suggested Value
pop	Population Size	32
$maxfe$	Max Func. Evals (each jump)	5,000
p_c	Probability of Crossover	0.9
p_m	Probability of Mutation	5/145
η_c	Distribution index for SBX	10
η_m	Distr. index for Polynomial Mutation	10

NSGA-II in Stage-3, such improvements are possible. What happens thereafter is remarkable!

There are 32 different solutions with a maximum constraint violation of 0.5% providing a saving of y_1-objective value of about 0.8 kg. If 0.5% relaxation is too large, a 0.1% relaxation of constraint bounds can bring in a saving of weight of about 0.35 kg, or a 0.2% relaxation can bring a saving of about 0.5 kg. The shape of this trade-off relationship between constraint bound relaxation and corresponding maximum saving in y_1 can help designers to set a suitable limit on relaxation of constraint bounds and then engage in negotiations with other concerned domain experts for a compromise solution.

For the two solutions having $y_1 = 168.78$ and 168.93 (shown in Figure 6(a)), Stage-3 is repeated. Figures 8 show the respective bi-objective trade-off solutions. Interestingly, in both cases, a large gain in y_1-objective is possible without any sacrifice of the constraint satisfaction. Thus, in addition to finding trade-off solutions between constraint relaxation and objective gain, Stage-3 also acts as an additional local search operator in the vicinity of the chosen Stage-1 solution. Similar trade-off behavior in both these cases can also be observed from the figures.

10 Conclusions

This paper has developed a three-stage optimizer, MOTRAN, to solve a large-scale, industry-standard, black-box optimization problem. The first stage of MOTRAN performs a global six-objective optimization and finds a set of diverse design space solutions emphasizing one of the critical objectives within the allowed computational budget. Stage-1 has used a gradient based local search method to locally improve the current best solution found from the population based search. This eventually results in a diverse design choices to the user. Stage-2 of MOTRAN has identified a pair of designs having a similar value of one of the objectives, while maintaining a wide difference in their variable values. Such diverse, multi-modal optimized solutions provide users flexibility with alternative good designs and the knowledge they commonly exhibit. Stage-3 of MOTRAN has been designed to perform a bi-objective neighborhood search near a chosen Stage-2 solution to find a set of new solutions trading-off constraint relaxation and objective improvement. Stage-3 of MOTRAN also provides other important information about the solution, such as: i) the number of constraints violated by each solution, ii) A set of violated constraints and their respective violation amount. This allows the user to have a sensitivity analysis around the chosen solution before making a final decision. Such information will provide a plethora of knowledge about the search space in the vicinity of the Stage-1 or Stage-2 solution(s) to explore the possibility of relaxing constraint values that

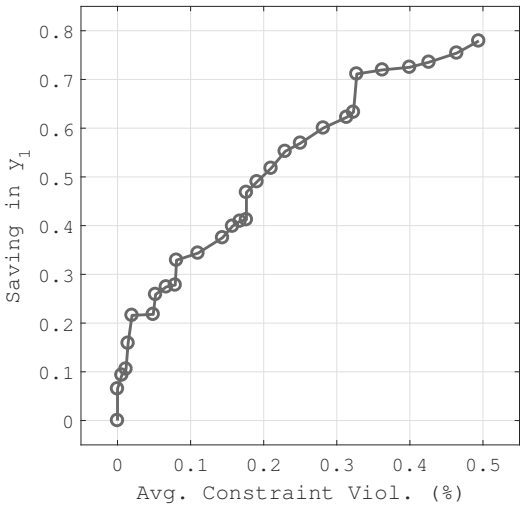


Figure 7: Saving in objective y_1 versus average constraint violation shown for the solution with lowest y_1 value, ($y_1 = 166.64$) as shown in Figure 4.

might offer better objective solutions. Moreover, this might also be the case the slight infeasibility could lead to a large reduction in the objective value, a fact which was not possible to comprehend without such a sensitivity analysis.

This study has utilized different optimization solution principles – multi-objective, multi-modal, highly constrained, low budget, discrete, gradient-based, population-based, and sensitivity-based search – to constitute new, unconventional, and unacademic optimization procedures for addressing various industry needs. Different problem types demanded a flexible yet efficient framework for optimization, which were enabled by the population-based evolutionary optimization framework. The tricks and methodologies used in this study should motivate researchers and stay as examples for integrating different optimization solution ideas to solve other similar or more challenging industrial problems.

Acknowledgment

This research was supported by General Motors. The views presented here do not necessarily reflect those of the sponsor whose support is gratefully acknowledged.

References

[1] A. Ahrari, A. A. Atai, and K. Deb. Simultaneous topology, shape, and size optimization of truss structures by fully stressed design based on evolution strategy (FSD-ES). *Engineering Optimization*, in press.

[2] Alberto Colorni, Marco Dorigo, and Vittorio Maniezzo. Genetic algorithms and highly constrained problems: The time-table case. In *International Conference on Parallel Problem Solving from Nature*, pages 55–59. Springer, 1990.

[3] D. W. Corne, J. D. Knowles, and M. Oates. The Pareto envelope-based selection algorithm for multiobjective optimization. In *Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature VI (PPSN-VI)*, pages 839–848, 2000.

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

[5] K. Deb. *Multi-objective optimization using evolutionary algorithms*. Wiley, Chichester, UK, 2001.

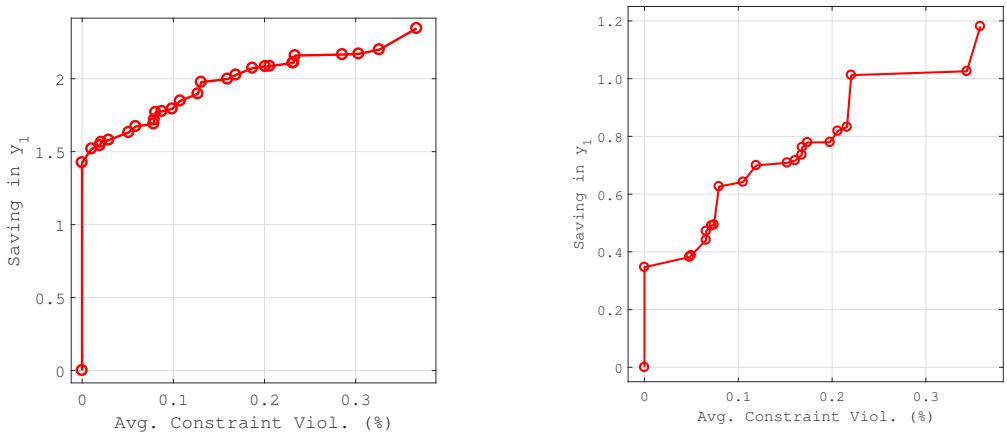


Figure 8: Saving in objective y_1 versus average constraint violation shown for the two niched solutions in Figure 6(a).

[6] K. Deb and M. Abouhawwash. A optimality theory based proximity measure for set based multi-objective optimization. *IEEE Transactions on Evolutionary Computation*, 20(4):515–528, 2016.

[7] K. Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9(2):115–148, 1995.

[8] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[9] K. Deb and R. Datta. Hybrid evolutionary multi-objective optimization and analysis of machining operations. *Engineering Optimization*, 44(6):685–706, 2012.

[10] K. Deb and D. E. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, 1989.

[11] K. Deb and 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):577–601, 2014.

[12] K. Deb and C. Myburgh. A population-based fast algorithm for a billion-dimensional resource allocation problem with integer variables. *European Journal of Operational Research*, 261(2):460–474, 2017.

[13] Juan J Durillo and Antonio J Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011.

[14] D. E. Goldberg. *Genetic Algorithms for Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[15] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pages 41–49, 1987.

[16] D. Hadka et al. Moea framework, 2014.

[17] Abdelaziz Hammache, Marzouk Benali, and Francois Aube. Multi-objective self-adaptive algorithm for highly constrained problems: Novel method and applications. *Applied Energy*, 87(8):2467–2478, 2010.

[18] H. Jain and 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):602–622, 2014.

[19] Muzaffer Kapanoglu and Ilker Ozan Koc. A multi-population parallel genetic algorithm for highly constrained continuous galvanizing line scheduling. In *International Workshop on Hybrid Metaheuristics*, pages 28–41. Springer, 2006.

[20] Michael Kokkolaras, Charles Audet, and John E Dennis Jr. Mixed variable optimization of the number and composition of heat intercepts in a thermal insulation system. *Optimization and Engineering*, 2(1):5–29, 2001.

[21] X. Li, M. G. Epitropakis, K. Deb, and A. Engelbrecht. Seeking multiple solutions: An updated survey on niching methods and their applications. *IEEE Transactions on Evolutionary Computation*, 21(4):518–538, 2017.

[22] K. Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer, Boston, 1999.

[23] Roberto Piola. Evolutionary solutions to a highly constrained combinatorial problem. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 445–449. IEEE, 1994.

[24] P. Reed and D. Hadka, 2017. Version 2.12.

[25] G. V. Reklaitis, A. Ravindran, and K. M. Ragsdell. *Engineering Optimization Methods and Applications*. New York : Wiley, 1983.

[26] Thomas P. Runarsson and Xin Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on evolutionary computation*, 4(3):284–294, 2000.

[27] András Sóbester, Alexander IJ Forrester, David JJ Toal, Es Tresidder, and Simon Tucker. Engineering design applications of surrogate-assisted optimization techniques. *Optimization and Engineering*, 15(1):243–265, 2014.

[28] S. Tiwari, P. Koch, G. Fadel, and K. Deb. Amga: an archive-based micro genetic algorithm for multi-objective optimization. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 729–736. ACM, 2008.

[29] Sander van Rijn, Michael Emmerich, Edgar Reehuis, and Thomas Bäck. Optimizing highly constrained truck loadings using a self-adaptive genetic algorithm. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 227–234. IEEE, 2015.

[30] Jonathan Wight and Yi Zhang. An ageing operator and its use in the highly constrained topological optimization of hvac system design. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 2075–2082. ACM, 2005.

[31] Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.

[32] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization. In K. C. Giannakoglou et al., editor, *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, pages 95–100. International Center for Numerical Methods in Engineering (CIMNE), 2001.

[33] G. Zoutendijk. *Methods of Feasible Directions*. Amsterdam: Elsevier, 1960.