Finding Near-Optimum and Diverse Solutions for a Large-Scale Engineering Design Problem

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

***Abstract*—Optimization problems in real-world usually involve a large number of variables and constraints. Moreover, practitioners also need to find a reasonable solution quickly, despite the fact that solution evaluation can be expensive. In addressing such optimization tasks, practitioners often have a completely different set of objectives than the usual trend of finding a single and accurate optimal solution. First, with the limited computational time availability and large dimension, it is not accepted (and in most occasions, not possible) to find the exact optimum solution of the problem, rather the focus must be spent to identify a near-optimal solution in a quick computational time. Second, instead of concentrating on a single solution, practitioners are often interested in evaluating a diverse set of solutions from the search space having similar objective values. Third, when a solution is finally chosen, practitioners are interested in exploring its vicinity to understand the trade-off between objective value and constraint violation, so that a more informed description of the problem can be ascertained. We highlight all these practicalities through a large-scale engineering design problem and present a suitable evolutionary-cum-classical, single-cum-multi-objective optimization method. The methods are generic and can be applied to other complex optimization tasks.**

***Index Terms*—Near-optimal solution, diverse solutions, multi-objective optimization, local Search, large-scale optimization, black-box optimization.**

I. Introduction

Optimization methods are increasingly being adopted in practice due to their ability to address a number of practicalities: (i) discrete search space, (ii) multiple conflicting objectives, (iii) a large number of variables, (iv) a large number of constraints, (v) black-box nature of problems, (vi) need for finding multiple diverse solutions, (vii) computationally expensive problems, (viii) a fixed number of solution evaluations, and others. In addressing these issues, deterministic and point-based optimization methods have shown difficulties, particularly in negotiating large dimensions and finding multiple diverse solutions in a multi-objective scenario. However, they have been demonstrated to be excellent local search methods for exploring a small neighborhood in the search space. On the other hand, evolutionary population-based optimization methods have been shown to have a more global perspective and are apt in finding diverse solutions. Due to their direct approach, they are also ideal for black-box optimization and subjected to an easy change in their operators to suit a problem.

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}@msu.edu, kdeb@egr.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

While an optimization algorithm is expected to find the true optimum solution of a problem, it is often far from possible, mainly due to limited number of solution evaluations that are possible to achieve within a stipulated amount of computational time allocated to solve a problem. Thus, in solving such real-world industrial problems, one aspect is clear: the objective is to find a near-optimum solution as close to the true optimum as possible. While a good solution is found, practitioners also expect to know if there exist any further solutions having a similar objective value. Although this may not provide a better solution, practitioners are often interested in this extra knowledge of additional good solutions, so the market or other situations can drive them to switch from one good solution to another. Finally, before a solution is finalized for implementation, practitioners are interested in knowing *sensitivity* of their chosen solution against constraint violation. Constraints have a lower or upper bound which are usually set by respective groups with an industry. They are not usually set in stone and are liable to change, may not be by a large amount, but surely slightly. If a relaxation of a specific constraint bound by a small acceptable amount can result in a significant improvement in the objective value in the vicinity of the chosen solution, practitioners would like to harness this fact with the respective group. Thus, a typical industry-standard optimization task may start with a global optimization task 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.

In this paper, we devise a two-stage hybrid optimization method to address each of the above issues by combining point and population-based optimization methods together. The working of the methods is demonstrated on a specific industrial problem having hundreds of variables and constraints. The problem is treated as a black-box problem, as no specific knowledge of the variables, constraints or objectives is used in the algorithms. The purpose of each stage and the obtained results are presented. The proposed two-stage method is generic and is ready to be applied to solve other complex industrial problems.

II. The Target Black-box Optimization Problem

The main goal of this paper is to solve a large-scale real-world engineering design optimization problem required by the GM Vehicle Optimization Group. The exact description of the problem is irrelevant since it was presented to the authors as a Black-box optimization problem. For the obvious reasons, the target optimization problem is very different from the canonical benchmark Multi-objective Optimization Problems (MOP) in terms of the number of variables, constraints and objectives. In terms of an actual implementation, the black-box function evaluates the objective value as "responses" from a given set of vector values (i.e. design variables). Here we point out the specific characteristics of the target optimization problem:

- The objective function values and the constraints are known in the form of response surfaces evaluated using an executable computer program. The output from this program can only be probed by a limited number of invocations.
- The problem requires 145 variable values $\mathbf{x} = (x_1, x_2, x_3, \cdots, x_{145})$ and computes 147 responses $\mathbf{y} = (y_1, y_2, y_3, \cdots, y_{147})$ as output.
- Each of the variable values $(x_1, x_2, x_3, \cdots, x_{145})$ have their respective lower and upper bounds, i.e. $(\{x_1^L, x_1^U\}, \{x_2^L, x_2^U\}, \cdots, \{x_{145}^L, x_{145}^U\})$, and each of them are discretized in 0.05 intervals.
- The responses $(y_1, y_6, y_{14}, y_{108})$ are to be minimized, and the responses (y_{29}, y_{146}) are to be maximized.
- Except for the first response y_1, we are free to choose any or all of them either as constraints or as objectives. In this experiment, we chose $(y_6, y_{14}, y_{108}, y_{29}, y_{146})$ as objectives. This comes from the fact that the response y_1 is more important than the rest and the search algorithm should not sacrifice y_1 by gaining more in other responses. Therefore, we call $(y_6, y_{14}, y_{108}, y_{29}, y_{146})$ as *helper objectives*.
- The responses $\mathbf{y}$ may have upper or lower bounds – but not both.
- Except for the response y_1, which is the primary objective in this problem, the rest of 146 responses all have bounds and therefore they are also constraints.
- A very narrow feasible region (146 inequality constraints, mostly active), such regions might appear repetitively throughout the whole search space.
- The gradient (i.e. Jacobian) of the objective space and the constraint space are also known and usable during the optimization run.
- Optima needs to be achieved within a limited computational budget, i.e 50,000 Solution Evaluations (SEs)[1], which is essentially the total number of invocations of the executable to compute the responses.
- A limit on the wall clock time available for one optimization run is $\approx$ 1-2 hours.

The extent of the problem difficulty is apparent from the above description. The main challenge comes from the fact that we do not have a clear understanding of the innards that constitutes the response surface models. Another challenging part of this research is the lack of examples in the multi-objective optimization literature that will suggest a streamlined approach for handling all the complexities described above, simultaneously. The next section discusses some useful ideas that helped us to formulate a pragmatic "plan of attack" for the solution.

[1]Here, we divert from the standard definition of *Function Evaluation (FEs)* since the computation of "one" objective value incurs a multiple number of actual mathematical function evaluations.

A. More Specific Requirements

The target black-box optimization problem also comes with a set of very specific requirements. Moreover, from the previous on-going investigations and the domain knowledge of the problem, we were given with only two instances of the sub-optimal solutions that our algorithm can utilize. We call these two points as $\mathbf{u}^*$ and $\mathbf{v}^*$. Further specific requirements to this problem is summarized as follows:

- **Convergence:** our algorithm needs to find significantly improved solutions from the given known points, i.e. $\mathbf{u}^*$ and $\mathbf{v}^*$.
- **Pareto-optimality:** we need to find a set of Pareto-optimal solutions that present trade-offs in the aforementioned 6 objectives.
- **Niche/Diversity:** our algorithm needs to find a set of Pareto-optimal solutions with similar (or almost equal) objective values with a completely different design variables.

III. Literature Survey

The algorithm (more appropriately, the solution framework) that we want to use is a population based stochastic search algorithm. In this section, we will address each of the problem characteristics and will try to rationalize our choice of algorithm and relevant modifications in the light of how they have been investigated by the Evolutionary Multi-objective Optimization (EMO) community.

A. More Than Three Objectives

For many-objective optimization problems (i.e. more than 3 objectives), MOEA/D and NSGA-III [1], [2] are viable choices. Both algorithms take advantage of decomposing the many objective optimization problem into a number of single objective optimization subproblems and optimize them simultaneously. Therefore, our framework exclusively depends on the NSGA-III algorithm.

B. A Small Feasible Region

As we have described earlier, the problem has 146 constraints, most of them being active during an optimization run. The strategy to handle such a constrained search-space is one of the most important parts of our algorithm. Problems with highly constrained search space can be found in the evolutionary optimization literature [3]–[8]. Among these, [4] treats the feasible and infeasible solutions in separate archives and claims to outperform NSGA-II in one such problem from the industry. Upon initial testing, the first author finds the results to be not very useful for our particular black-box problem. On the other hand, [8] encourages the use of a new parameter called *aging* of genes in order to avoid domination of initial population by a very few feasible solutions and introduce diversity in the population. [7] deals with a highly constrained combinatorial single objective optimization problem and utilizes a problem specific encoding scheme. [5] uses a Multi-population Parallel GA along with a repairing mechanism to reduce all violations in the best solution one by one. Another idea of tackling highly infeasible regions can be of using the idea of *stochastic ranking* by [9] that aims to balance the dominance of penalty and objective functions. The method aims at reducing infeasibility in the beginning and reducing the function value in the later stages of optimization.

Unlike our target problem, in all of the aforementioned works, the optimization problem was neither a black box, nor the design space was of mixed variable type. Problems with only real variables employ help of classical techniques, where as problems with (only) discrete variables employ very problem specific crossover and mutation operators. Moreover, as we have mentioned before, the requirement of this project is to find a multiple trade-off solutions from the different niches, such that two significantly different solutions will have a similar (or almost same) objective values. As a result, none of the existing methods that have been already experimented in the literature is directly applicable to our scenario. Furthermore, in the later sections, we will see that the black box nature of the problem necessitates the use of some ad-hoc or customized local-search operations to aid the convergence in a significant factor.

C. Limited Computational Budget

One of the key requirements to this investigation is to find solutions within a strict computational budget. In our experiments, the computational budget is primarily defined in terms of wall clock time of 1-2 hours. To be more specific, if our algorithm is performed on a single core computer[2], this roughly translates into 60,000 Solution Evaluations (SEs).

Such a constricted budget makes the overall task very difficult if we solely depend on the random mutation and crossover operations. As a result, without some sort of deterministic search strategy, achieving a fast convergence is extremely challenging, even if some known solutions (i.e. $\mathbf{u}^*, \mathbf{v}^*$) are already provided. Therefore, our framework exclusively utilize the gradient information to alter the promising solutions from the current population to explore new niches in the search space. Also our method builds up an archive of good and already explored solutions so that local optima can be avoided during the run. This local search operation is the key mechanism of our algorithm that helps to explore the different small feasible regions inferred by the diversities that come from the population members.

D. Discrete Variables

Discrete variables present one more challenge in reaching optimal solutions as they pose difficulty in neighborhood prediction to classical and evolutionary methods alike. Evolutionary algorithms perform relatively better in current scenario when the problem is a black box and there is a large number of design variables. One strategy can be to use problem specific crossover and mutation operators to get out of bad regions and of course archiving solutions helps in not loosing good solutions. However, developing any such operator will require a *notion of a neighborhood* on the discrete variables. This is a difficult task in case of categorical variables. [10] shows developing a sound neighborhood notion for categorical variables in an engineering design problem. It requires some knowledge of the underlying nature of the search space. However, our local search algorithm does not depend on such neighborhood constructs.

[2]3.6 GHz Intel Core-i7, 12 GB RAM, 64bit Linux

IV. The Proposed Methodology: MOTRAN

Keeping in mind all of the limitations discussed in the previous section, our proposed framework will minimize a total number of 4 responses: y_1, y_6, y_{14} and y_{108}; and maximize 2 responses: y_{29} and y_{146}. We call our framework as *Multi-objective Trade-off Analyzer* or *MOTRAN* in short. The basic work flow of MOTRAN is composed of two stages:

- **Stage-1:** Stage-1 aims to provide a set of diverse solutions (in design space) that are close in the y_1 objective. This stage will utilize a "many objective optimization" algorithm called NSGA-III [1] to optimize all 6 objectives indicated earlier. This stage should also employ some local search and niching strategies.
- **Stage-2:** Stage-2 provides information on constraint violation versus the improvement in response y_1 in the neighborhood of a user specified solution from stage-1. The Decision Maker (DM) can choose a particular solution according to the design requirements and study the critical constraints hindering the improvement of a solution in response y_1.

It should be noted that the stage-1 takes $\sim 30k$ SEs ($\sim$1 hrs) and the stage-2 takes $\sim 5k - 10k$ evaluations (0.25-0.5 hrs), totaling in 1.5 hrs on our machine[2]. Moreover, as the black-box function is provided as a static object (i.e. `.so` file), the individual function calls may be slower. If the black-box is completely integrated into MOTRAN, surely things will move faster. For this reason, we show results in terms of SEs and not in terms of the wall clock time. Since the native MOTRAN has been developed in Java, the only bottleneck seems to be the invocation of the static library for the function evaluations. In order to understand the methodology behind MOTRAN, first we need to analyze the given black-box problem in a more structured way. In order to do this, we have done a large scale Latin Hypercube (LHC) sampling with *2.5 million* solutions and there was not even a single feasible solution found (Figure 1). Only 17 constraints (of 146) were satisfied by all solutions. Therefore, the first goal was to get an overall idea about the structure of the underlying search space.

A. Search space: how might it look like?

For this part of the experiments, we have run a population based stochastic optimization algorithms (considering only one objective y_1) with varying parameters, the goal of this experiment was to infer how the search space might look like. The runs were conducted using the two-objective NSGA-II [11] and a simple real-parameter GA. In the NSGA-II experiment, we took y_1 as one objective and the constraint

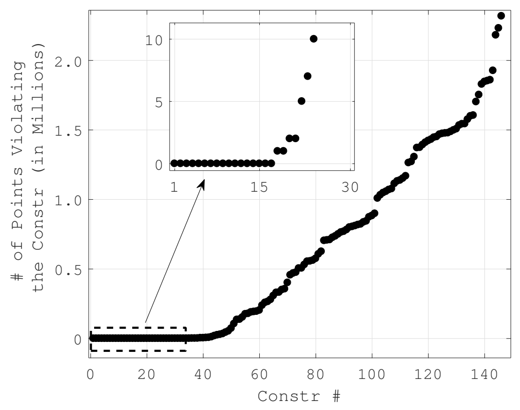

Fig. 1. 2.5 million Latin hypercube samples. Here the x-axis is the response id's and the y-axis indicates the number of infeasible solutions.

violation (with a tolerance of 0.1%) as another objective, for the GA case y_1 was considered as one objective.

The real-parameter GA [12] and NSGA-II (with constraint violation as second objective) both are able to reach the region R_2 fairly quickly ($<$ 1000 solution evaluations). The search algorithm can reach the region R_3 in 50k SEs. What we think that beyond this point, the feasible space become very narrow. As a result, when a stochastic algorithm applies a random local search (i.e. mutation), the solutions start to violate one of the 146 constraints. Thus making it very difficult to reach the region R_4, which is the best known optima (i.e. $\mathbf{v}^*$). In the next section, we discuss the basic architecture of MOTRAN in more details. A sketch describing both stages is shown in Figure 2.

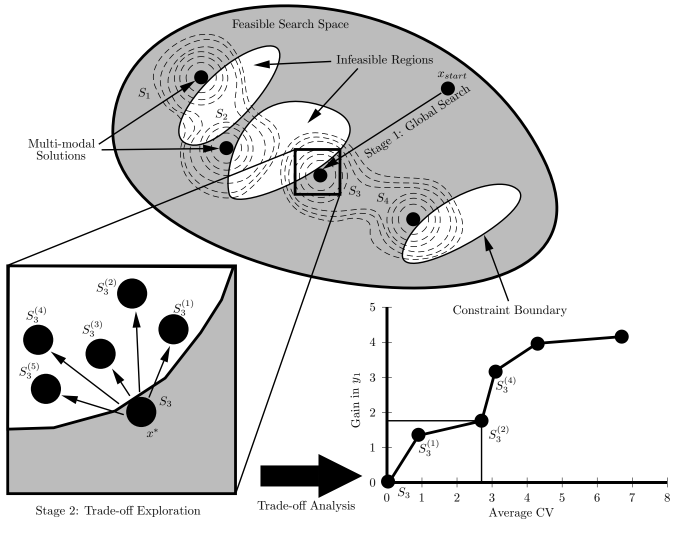

Fig. 2. The two stage mechanism of MOTRAN. In the Stage-1, it will start from the solution $\mathbf{u}^*$ and run a global search to reach a solution $\mathbf{x}^*$.

V. Stage-1: Many-objective Global Search Aided by Local Search

As we have mentioned before, this stage conducts the primary global search to find a set of Pareto-optimal solutions that are:

- diverse in design space,
- in an acceptable range of y_1 values.

A six-objective problem is solved in this stage using a variant of NSGA-III algorithm. In most cases where the optimization problems have constraints, the global minimum lies on the boundary of the constraints, which is also indicated in the Figure 2. Moreover, since the problem has 146 constraints, there is a good possibility that the search space contains multiple number of feasible 'pockets' with similar objective values. Therefore, during the Stage-1, the MOTRAN framework will formulate a six-objective optimization solver that facilitates diversity within a small population size. However, such 'exploration' mechanism will not suffice a successful search. In order to promote 'exploitation', we have also devised a very efficient Local Search (LS) algorithm that will be used by the Stage-1 of MOTRAN. The LS algorithm is a *Feasible Steepest Descent Search*, which is customized to work over the discrete variable space. A detailed explanation of this algorithm is discussed in the Section V-A. As LS can find an improved solution relatively quickly from a given point (although such descent ends at some point), we also add the best known solution ($y_1(\mathbf{v}^*) = 168.765$) inside the Stage-1 initial population. For the Stage-1, we set a limit of 30k SEs.

A. Local Search : Feasible Steepest Descent Search

This search mechanism is an important part of MOTRAN Stage-1. Because of large number of constraints, exploration with random mutation and crossover is very difficult in maintaining solution feasibility. We have implemented this algorithm to ensure a deterministic improvement on the objective y_1. Although there are other objectives to be optimized, the main goal of this project is get a set of minimal y_1 solutions with wide range of trade-offs over y_6, y_{14}, y_{108}, y_{29} and y_{146}. This should also noted that the Jacobian $\mathbf{J}$ of the responses, are known. For the obvious reason, $\dim(\mathbf{J}) = m \times n = 147 \times 145$. Given a current solution $\mathbf{x} = (x_0, x_1, \ldots, x_n)$, all responses $\mathbf{y} = (y_0(\mathbf{x}), y_1(\mathbf{x}), \ldots, y_m(\mathbf{x}))$ and $\mathbf{J}$ evaluated at $\mathbf{x}$. The local search algorithm works as follows:

- *Step 1*: Assume the current solution to be $\mathbf{x}^{(t)} \leftarrow \mathbf{x}$, therefore $\mathbf{y}^{(t)} \leftarrow \mathbf{y}$.
- *Step 2*: Evaluate normalized constraint violation $\mathbf{c}(\mathbf{y}^{(t)}) = (c(y_0^{(t)}), c(y_1^{(t)}), \ldots, c(y_m^{(t)}))$ without tolerance and normalized constraint violation $\hat{\mathbf{c}}(\mathbf{y}^{(t)}) = (\hat{c}(y_0^{(t)}), \hat{c}(y_1^{(t)}), \ldots, \hat{c}(y_m^{(t)}))$ with tolerance $\varepsilon = 0.001$.
- *Step 3*: According to the sign of $\mathbf{J}_{0j}$, find the possible directions of changes in $\mathbf{x}^{(t)}$, $\Delta\mathbf{x}^{(t)} = (\Delta x_0^{(t)}, \Delta x_1^{(t)}, \ldots, \Delta x_n^{(t)})$ such that none of them crosses the respective variable bound.
- *Step 4*: If $\Delta\mathbf{x}^{(t)}$ is a zero vector, then no improvement is possible, so $\mathbf{x}$ does not change. Otherwise, compute $\Delta\mathbf{y} = \mathbf{J} \circ \Delta\mathbf{x}^{(t)}$ where $\circ$ is a row-wise *Hadamard product* of $\mathbf{J}$ and $\Delta\mathbf{x}^{(t)}$, i.e. $\Delta\mathbf{y}_{ij} = \mathbf{J}_{ij}\Delta\mathbf{x}_j^{(t)} = \frac{\partial y_i^{(t)}}{\partial x_j^{(t)}}\Delta x_j^{(t)}$.
- *Step 5*: Compute estimated y_0 for all possible Δx_i's, $\hat{\mathbf{y}}_0 = (y_0^{(t)} + \Delta\mathbf{y}_{00}, y_0^{(t)} + \Delta\mathbf{y}_{01}, \ldots, y_0^{(t)} + \Delta\mathbf{y}_{0n})$.
- *Step 6*: Now go through each column j of $\Delta\mathbf{y}$ and compute the sum of all normalized estimated gain of each responses

w.r.t. to the corresponding variable change Δx_j, i.e. $\Delta \hat{\mathbf{g}}_j = \sum_i (\hat{c}(y_i^{(t)}) + \Delta \mathbf{y}_{ij}) - \varepsilon$ and $|\Delta \hat{\mathbf{g}}_j| = n = 145$.

- *Step 7*: Now from $\Delta \hat{\mathbf{y}}_j$, find all the indices $j \in I$ such that $\Delta \hat{\mathbf{g}}_j = 0$ and $\hat{y}_j < y_0^{(t)}$, where $\hat{y}_j \in \hat{\mathbf{y}}_0$. Sort the indices I with respect to $\hat{y}_j$.
- *Step 8*: Create a reduced set of indices $I_r \subseteq I$ such that I_r does not contain an index i if there already an index $j \in I_r$, given that $||\hat{y}_i - \hat{y}_j|| < \varepsilon_{y_0}$. Where ε_{y_0} is the threshold radius of estimated gain ($\varepsilon_{y_0} = 0.001$) that does not include two close values from $\hat{\mathbf{y}}_0$.
- *Step 9*: Now go through each of the indices $i \in I_r$ and apply $x_i^{(t+1)} \leftarrow x_i^{(t)} + \Delta x_i^{(t)}$ and evaluate $y_0^{(t+1)}(\mathbf{x}^{(t+1)})$. If $y_0^{(t+1)} < y_0^{(t)}$, then go to next step. Otherwise, there is no way to make improvement and $\mathbf{x}^{(t)}$ is the optimized solution.
- *Step 10*: Jump to *Step 1* and proceed to next iteration, $t \leftarrow t+1$.

The above algorithm is applied to the all feasible solutions in the population during a NSGA-III run. Furthermore, the starting solution, the final solution and all intermediate solutions of a local search are archived and tagged so that any future local search from these points is carried out only once in an NSGA-III run. A next local search operation will only be initiated if a new feasible solution is found from a mutation or crossover operation. For example if we start our local search from the best known solution ($y_1(\mathbf{v}^*) = 168.765$), we can reach to a solution with $y_1 = 167.152$ in 94 function evaluations. Similar to tabu search, all solutions with no further improvement by the local search are tagged as "exhausted" and are not exposed to further local search operation in the future generations.

As can be seen in Figure 3, the solution with $y_1 = 167.152$ does not differ much from the $y_1 = 168.765$ solution in the design space. Hence, *our local search algorithm can improve a solution only in close vicinity of the (local) optima*. Therefore, a global search algorithm like NSGA-III, along with a custom niching strategy, is needed to find the different niches in the design space that may yield either:

- a solution with better y_1 response or,
- a solution within an acceptable range of y_1, but that is very different in the design space.

VI. Other Important Capabilities

The goal of this investigation is to develop a technique that can find multiple trade-off solutions within an acceptable ranges from the best known solutions $\mathbf{u}^*$ and $\mathbf{v}^*$. Also with a significantly different values in the variable space. For this, we need to consider some more extra mechanisms to ensure requirement of multi-modal/niched solutions. In the following subsections, we present the modifications performed on NSGA-III procedure for our stage-I task.

A. Niche Exploration in The Variable Space

If we look at the *Step 9* above, at each iteration we will have a new solution $\mathbf{x}^{(t+1)}$. Since this solution is found by changing one variable from the previous iteration, this solution

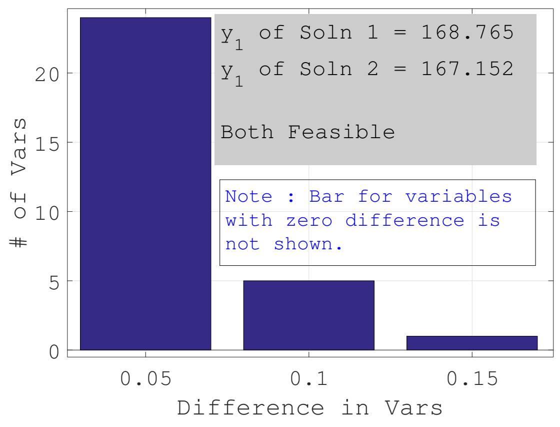

Fig. 3. The bar graph shows the difference between the $y_1 = 168.765$ solution and the solution obtained after applying the local search to reach the $y_1 = 168.765$ solution.

is different from $\mathbf{x}^{(t)}$ and also it has a better y_1 value than the previous one. Moreover, we might get two solutions with similar function values if the local search operates from two different starting points. As a result, $\mathbf{x}^{(t)}$ and $\mathbf{x}^{(t+1)}$ might belong to two different niches. In order to find such multiple niches and to facilitate the local search within those, we keep all solutions that were found from the *Step 9* above in an *archive*. The solutions from the archive are included in the NSGA-III's merge operation. The global search capability of NSGA-III equipped with a niching strategy (explained ahead) utilizes the solutions from the archive and tries to maintain separate niches in the design space. At the same time, the local search described in Section V-A capitalizes on locally optimizing newly found niches.

B. Maintenance of The Explored Niches

NSGA-III assumes a set of reference directions in the objective space, and tries to maintain at least one solution along each reference direction. In order to ensure that currently preferred solutions are emphasized, during the merge operation, we need to conduct a careful direction re-assignment. This re-assignment protects N solutions, that are not only the best in y_1 (the most important objective function of the problem) but also separated in design space by some *niche-radius* 'ϵ_x'. This helps the solutions from getting lost and replaces the directions that are not associated with any individual or are associated with worse solutions in y_1. During the elite selection in NSGA-III, the merged population is sorted with respect to y_1. For obvious reason, one of the reference directions must protect the best solution in y_1, therefore, at least one reference direction is re-assigned. After this, from the best y_1 solution, we discard the solutions that are within a niche-radius of ϵ_x in the design space. This discard operation is necessary to keep the archive in a fixed size. As a result, if the number of such solutions in the archive is n, then each of these solutions yields n separate reference directions. Moreover, if the population size be n_{pop}, two cases may arise:

- $n \geq n_{\text{pop}}$: in this case, top n_{pop} number of newly generated reference directions will replace all the previous reference directions.
- $n < n_{\text{pop}}$: in this case, top n_{pop} number of newly generated reference directions replace n existing reference directions randomly.

C. Identifying Pair-wise Trade-off Solutions from Different Niches

Once the stage-1 optimization algorithm ends, the next step is to find a set of pair-wise solutions that belong to different niches. At this stage, the algorithm selects k-pairs of best possible solutions in y_1 (the most important objective) with different level of trade-offs in the other 6 objectives:

- a maximum difference in y_1, say Δy_1, within which two solutions can be considered close to each other in y_1 and,
- a minimum design space distance, say Δx, outside which the user considers two solutions to belong to two different niches,

In the PCP plot (Figures 4 and 5), $k = 3$ and the best y_1 solutions from the different niches are presented with three different colors. For example, in the case of the red pair of solutions, they differ in y_1 by only 0.15 but they are very different in the design space as the histogram of Figure 5a suggests.

VII. Stage-2: Trade-off Exploration Phase

In this stage, we consider each solution obtained by stage-1 and perform a local relaxation to understand the trade-off between the extent of constraint violation and improvement of the most important objective function. This is performed with the hope that in case a much better objective solution, albeit with a small constraint violation, exists in the neighborhood of a Pareto-optimal solution, the designers are interested in evaluating that solution before accepting the feasible Pareto-optimal solution.

This stage is presented in the two small figures in the bottom of Figure 2. The goal of this part is to find a set of trade-off solutions around the neighborhood of the best y_1 solution $\mathbf{x}^*$ and also the critical constraints and their violation levels. Given that we allow a specified amount of infeasibility in the vicinity of $\mathbf{x}^*$.

Given a tolerance on the Constraint Violation (CV) (i.e. %0.5), the DM is interested to see how one can decide among different trade-offs. Therefore, the MOTRAN will run another batch of neighborhood search from $\mathbf{x}^*$ and find a set of Pareto-optimal solutions. This is again a multi-objective search which considers y_1 and the average CV of all the responses (CV_{avg}) as two objectives. Where CV_{avg} is evaluated only with violated constraints. However, the local search is different from the Stage-1. Here we generate a set of initial solutions from $\mathbf{x}^*$ by changing each of the design variable value by a small discrete steps, that are a small multiples of 0.05. From these initial set of solutions, we run an EMO algorithm. At the end of the Stage-2 run, the DM will be provided with a set of Pareto-optimal solutions in terms of the response y_1 and the CV_{avg}. We set a total 10k SEs for this stage. The next section discusses the results obtained on the GM problem using MOTRAN and a start point with $y_1(\mathbf{u}^*) = 184.2$.

VIII. Results: Stage-1

The Stage-1 results are found from the many-objective NSGA-III runs.

A. Parameter Settings

Table I shows the parameter values for the Stage-1 optimization. Best known solutions with $y_1(\mathbf{u}^*) = 184.2$ and $y_1(\mathbf{v}^*) = 168.76$ are introduced in the starting population. Stage-1 optimizer with the parameter values summarized below consistently yielded a set of solutions from different niches. They range from $y_1 = 166.65$ to $y_1 \approx 175$ and takes 30,000 SEs (1 hour) to compute.

TABLE I
Parameter Suggestions 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	20
η_m	Distr. index for Polynomial Mutation	20
ctol	Constraint Tolerance (in %)	0.001 – 0.005

- We have found that Stage-1 optimizer performed better with a population of 84. The population size is mainly dictated by the specific way to set the reference directions, i.e. layering approach [1].
- The value for the probability of mutation, p_m, in the Stage-1 is $15/145$ where 145 is the variable length.
- The constraint tolerance, *ctol* is set from 0.1% to 0.5%. Very low constraint tolerance completely eliminates almost all infeasible solutions from competing with good solutions and hence there is no diversity left in the population. Whereas, allowing a large value for the tolerance allows too many infeasible solutions but they are spread all over the landscape resulting in a slower convergence.

B. Final Solutions

Now we present some results obtained by the Stage-1 optimizer. The parameter settings for the run are given in Table I with *ctol* $= 0.1$ %. The Figure 4 illustrates the set of Pareto-optimal solutions with best values in y_1 that lie in different niches. If we look at the two solutions represented by the red lines, we can see that they are very close in y_1, however they have a complete non-identical trade-offs in other objectives. The solutions representing in red, green and blue lines are illustrated in the Figure 5. The x-axis denotes the difference of variable values and the y axis denotes total number of places they differ. For example, in the red solutions, they have similar objective values (i.e. 168.93 and 168.78), but they differ in 0.2 distance at more than 8 locations. Therefore, this result proves that MOTRAN can generate a wide range of diverse solutions with trade-offs in 6 different objectives.

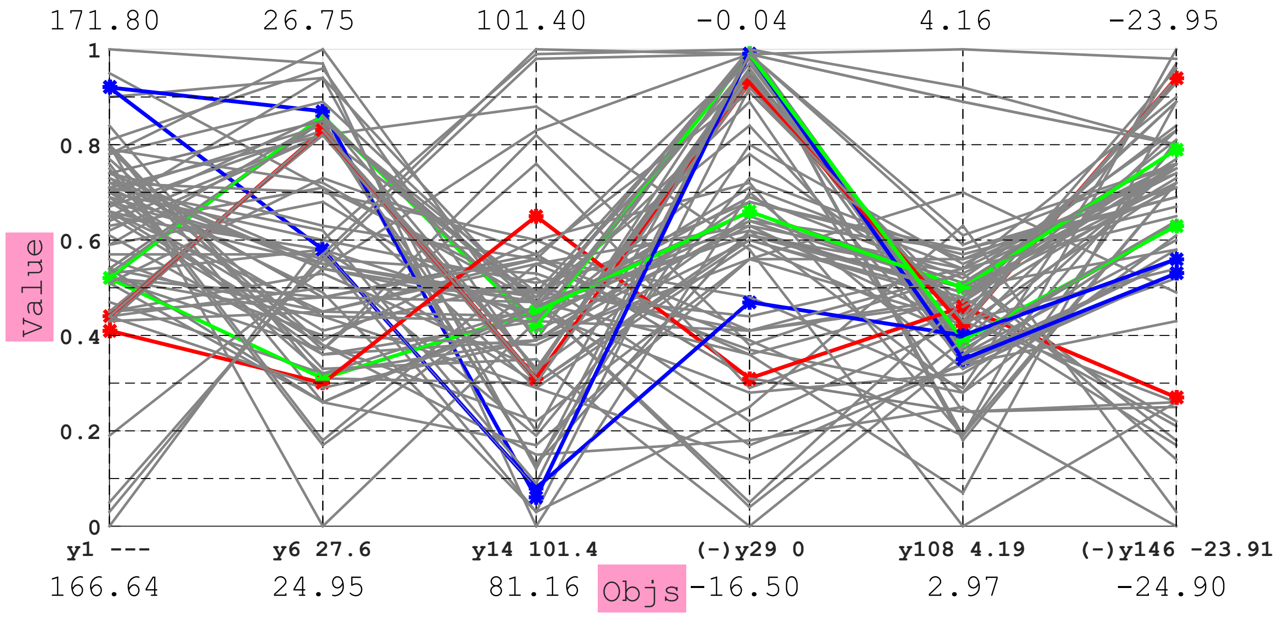

Fig. 4. Trade-off solutions found from the Stage-1 of MOTRAN.

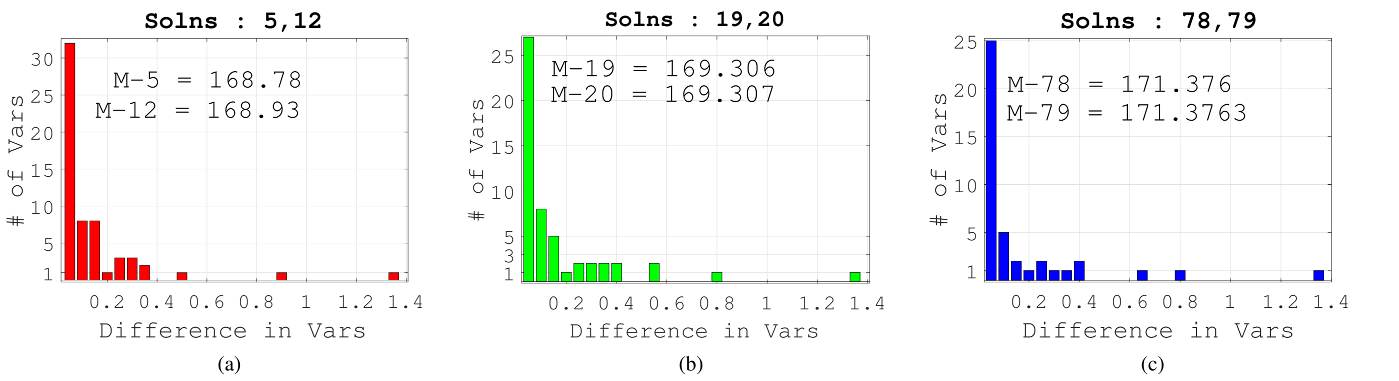

Fig. 5. The histogram of the variable values of the solutions corresponding to the red, green and blue lines in the PCP.

IX. Results: Stage-2

This section presents the results obtained from Stage-2 optimizer. The Stage-2 optimizer takes the best feasible point of Stage-1 as input. The best feasible solution found from the Stage-1 has a y_1 value of 166.64. Let's denote this solution as $\mathbf{x}^*$.

A. Recommended Parameter Settings

Table II shows the recommended parameter settings for the Stage-2 optimizer. For the local perturbation of $\mathbf{x}^*$, we pick one variable randomly and increase or decrease by 2 to 5 multiples of 0.05. Each multiple of 0.05 is called as δx. More specific settings are summarized as follows:

- 5,000 function evaluations are recommended for each δx jump from the best known solution of Stage-1. The user can conduct as many local jumps as the computational budget allows.
- A δx jump of 2 to 5 steps is recommended for conducting the local search by Stage-2 optimizer.
- As Stage-2 optimizer does not allow the individuals to move too far because of reduction in variable bounds around that of $\mathbf{x}^*$, a constraint violation tolerance of 100% is used in this case.

B. Trade-off Solutions from Stage-2

From the Pareto optimal front obtained at the end of Stage-2 optimizer by conducting a local jumps on the variables within a step size of 3 from $\mathbf{x}^*$, Figure 6 shows the points that have an average constraint violation of 2% or lower. Note that since a bi-objective optimization method is used in Stage-2, these Pareto-optimal solutions provide a better trade-off than

TABLE II
PARAMETER SUGGESTIONS FOR STAGE-2.

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
j	Step	2 to 5
ctol	Constraint Tolerance (in %)	100

any other random neighboring solutions. For example in the Figure 6, for an average constraint violation from $0\% \sim 0.5\%$, a maximum reduction in y_1 of 0.5 could be achieved from the best feasible solution. Stage-2 analysis not only provides a few alternate solutions having *maximum* reduction in y_1 for a small sacrifice of constraint violation, it also provides information of critical constraints and the ranking of their importance.

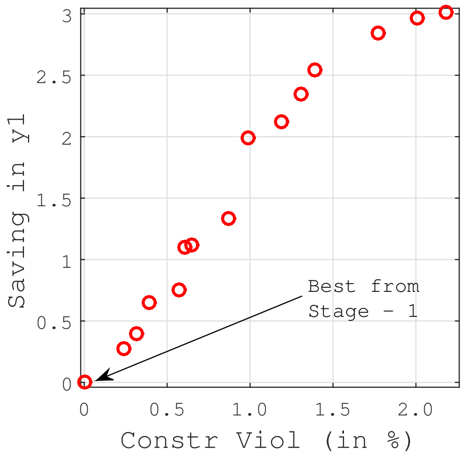

Fig. 6. Saving in y_1 versus constraint violation shown for the same points. Only points with Average Constraint violation less than 2% are shown here.

X. CONCLUSIONS

We have developed a two-stage optimizer, MOTRAN, to solve a large-scale, industry-standard, black-box optimization problem. The first stage of MOTRAN performs a global optimization and finds a set of diverse design space solutions that have similar value of the most significant objective by restricting the task within the allowed computational budget. Stage-1 also utilizes a feasible direction based steepest descent search mechanism to locally improve the current best solution found from the population based search. This eventually gives a diverse design choices to the user, where two designs with similar utility can be built from two very different sets of design parameters. Moreover, the parallel coordinate plot has shown the underlying trade-offs in all six objectives of the problem, when choosing a solution from the Pareto-optimal set.

The second stage of MOTRAN performs a local search near each of the chosen solutions ($\mathbf{x}^*$) and provides the user with trade-off information in the vicinity of each solution. Each of these solutions carries the potential of further improvement in the most significant objective y_1 along with an estimate of how much constraint violation that can be relaxed. A user can choose any one solution and can obtain valuable information on the trade-off between y_1 and the possible range of lee ways for constraint violation. Stage-2 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. Such information will provide a plethora of knowledge about the search space in the vicinity of the Stage-1 solution(s) to explore the possibility of relaxing constraint values that might offer better objective solutions. As a result, this provides valuable insights to alternative design choices. Moreover, this might also be the case that a slight infeasibility could lead to a maximum possible reduction in the objective value.

ACKNOWLEDGMENT

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

REFERENCES

[1] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2014.

[2] H. Jain and K. Deb, "An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part ii: handling constraints and extending to an adaptive approach," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 602–622, 2014.

[3] A. Colorni, M. Dorigo, and V. Maniezzo, "Genetic algorithms and highly constrained problems: The time-table case," in *International Conference on Parallel Problem Solving from Nature*. Springer, 1990, pp. 55–59.

[4] A. Hammache, M. Benali, and F. Aube, "Multi-objective self-adaptive algorithm for highly constrained problems: Novel method and applications," *Applied Energy*, vol. 87, no. 8, pp. 2467–2478, 2010.

[5] M. Kapanoglu and I. O. Koc, "A multi-population parallel genetic algorithm for highly constrained continuous galvanizing line scheduling," in *International Workshop on Hybrid Metaheuristics*. Springer, 2006, pp. 28–41.

[6] R. 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*. IEEE, 1994, pp. 445–449.

[7] S. van Rijn, M. Emmerich, E. Reehuis, and T. Bäck, "Optimizing highly constrained truck loadings using a self-adaptive genetic algorithm," in *2015 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2015, pp. 227–234.

[8] J. Wight and Y. 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*. ACM, 2005, pp. 2075–2082.

[9] T. P. Runarsson and X. Yao, "Stochastic ranking for constrained evolutionary optimization," *IEEE Transactions on evolutionary computation*, vol. 4, no. 3, pp. 284–294, 2000.

[10] M. Kokkolaras, C. Audet, and J. E. Dennis Jr, "Mixed variable optimization of the number and composition of heat intercepts in a thermal insulation system," *Optimization and Engineering*, vol. 2, no. 1, pp. 5–29, 2001.

[11] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[12] K. Deb, A. Anand, and D. Joshi, "A computationally efficient evolutionary algorithm for real-parameter optimization," *Evolutionary computation*, vol. 10, no. 4, pp. 371–395, 2002.