2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Multi-Scenario Optimization Using Multi-Criterion Methods: A Case Study on Byzantine Agreement Problem

Ling Zhu*, Kalyanmoy Deb*† and Sandeep Kulkarni*
*Department of Computer Science and Engineering
†Department of Electrical and Computer Engineering
Michigan State University
East Lansing, Michigan 48824
Email: {zhuling, kdeb, sandeep}@msu.edu

***Abstract*—In this paper, we address solution methodologies of an optimization problem under multiple scenarios. Often in practice, a problem needs to be considered for different scenarios, such as evaluating for different loading conditions, different blocks of data, multi-stage operations, etc. After reviewing various single-objective aggregate methods for handling objectives and constraints under multiple scenarios, we then suggest a multi-objective optimization approach for solving multi-scenario optimization problems. On a Byzantine agreement problem, we demonstrate the usefulness of the proposed multi-objective approach and explain the reasons for their superior behavior. The suggested procedure is generic and now awaits further applications to more challenging problems from engineering and computational fields.**

I. Introduction

In many engineering and computational optimization problems, a solution must be evaluated against a number of scenarios [1][2]. For example, in a structural optimization problem, a solution must usually be checked under a number of loading conditions arising from various considerations, such as from severe wind conditions providing lateral loads and from extreme vertical loads occurring from additional vehicular loads, heavy snow conditions, etc. In such problems, a solution is considered acceptable only if it performs in a satisfactory manner to not one but all specified loading scenarios.

Let us say that there are K different scenarios that must be considered for an optimization task:

$$\begin{array}{ll} \text{Minimize} & \sqcup_{k=1}^{K} f(k,\mathbf{x}), \\ \text{subject to} & \sqcup_{k=1}^{K} g_j(k,\mathbf{x}) \geq 0, \quad j=1,2,\ldots,J, \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)}, \quad i=1,2,\ldots,n. \end{array} \tag{1}$$

Here, the symbol $\sqcup$ signifies an aggregate performance of objective function $f()$ or constraint function $g_j()$ under all K scenarios. Figure 1 provides a sketch of the multi-scenario optimization problem. The evaluation of the objective function for a solution requires computation of it for all K scenarios and then derive an aggregate measure which can then be optimized. A solution $\mathbf{x}$ will be considered *feasible*, only if it satisfies each constraint for all K scenarios. In its traditional sense, if any constraint gets violated for any of the given scenarios, then the solution is considered infeasible.

In this paper, we address the solution of above problem using different aggregation methods for handling multi-scenario treatment of objectives and constraints used in practice. In doing so, some critical properties of resulting optimization task are revealed. Thereafter, a multi-objective approach is suggested to address the solution of the above problem. Although the paper discusses a generic methodology, we demonstrate the proposed methodology on a Byzantine agreement problem, which is of great importance in fault tolerant system design and in many military applications [3][4].

The rest of this paper is organized as follows: Section II presents different aggregation methods used for handling objective function and constraints. The proposed multi-objective methodology is described in Section III. Thereafter, in Section IV, we discuss the Byzantine agreement problem and the requirement for considering multiple scenarios in solving the problem. Section V presents the results using five different single and multi-objective methodologies. The reasons for superior performance of the multi-objective optimization approach is also discussed in Section VI. Finally, conclusions of this study are drawn in Section VII.

II. Methodologies for Handling Multiple Scenarios

In this section, we discuss different methodologies that are used for handling the objective function and constraints under multiple scenarios.

A. Handling the Objective Function

In the presence of multiple scenarios (S_k, $k = 1,2,\ldots,K$), the objective function must first be computed for each scenario, thereby computing $f(k,\mathbf{x})$ for every k. For example, in a structural optimization problem under multiple loading conditions, if the cost of fabricating a structure (denoted by a set of design variables $\mathbf{x}$) is the objective function, the fabrication cost must first be computed for each of the loading conditions. Several aggregation methodologies can be used in practice with the cost values obtained for each scenario.

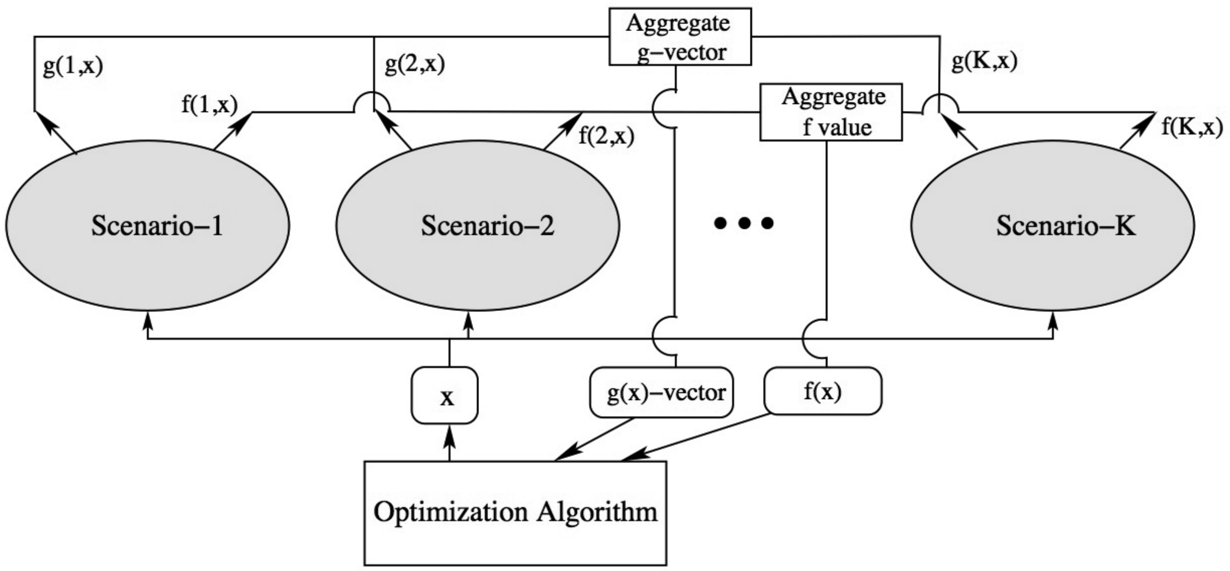

Fig. 1. An optimization algorithm must get a solution $\mathbf{x}$ evaluated for all scenarios and use an aggregate measure of constraints and objective function values to proceed.

1) Worst-Case Aggregation: A common strategy which is followed in practice is to find the worst cost of all scenarios and is used as the objective function of the optimization problem, that is, for minimization problems,

$$\sqcup_{k=1}^{K} f(k,\mathbf{x}) = \max_{k=1}^{K} f(k,\mathbf{x}). \tag{2}$$

Thus, for a minimization problem, the overall problem becomes a min-max problem. For maximization problems, the operator max should be replaced by a min operator, thereby having a max-min problem. This approach may provide a pessimistic estimate of the objective function, since the worst-case scenario may be an isolated event which may not represent the true performance over all K scenarios. The following approach can be used.

2) Average-Case Aggregation: An average objective value of all K scenarios can be used, instead:

$$\sqcup_{k=1}^{K} f(k,\mathbf{x}) = \mu_f^K(\mathbf{x}) = \frac{1}{K}\sum_{k=1}^{K} f(k,\mathbf{x}). \tag{3}$$

The average-case scenario is logical particularly when there is not much difference in performance among all K scenarios. When the variance in performance among K scenarios is large, a more statistically favorable aggregation would be better, such as the median-case or the following measure.

3) Mean-Variance Aggregation: Instead of an average, the mean and standard deviation of the objective function among all K scenarios can be used for minimization problems:

$$\sqcup_{k=1}^{K} f(k,\mathbf{x}) = \mu_f^K(\mathbf{x}) + \kappa\sigma_f^K(\mathbf{x}), \tag{4}$$

where $\sigma_f^K(\mathbf{x})$ is the standard deviation of $f()$ over K scenarios. The parameter κ can be chosen as 1, 2 or 3, depending on the importance of the standard deviation over the mean. Such an aggregate measure will give adequate importance to the distribution of $f()$ for different scenarios. For maximization problems, κ can be considered negative. Due to its complexity, we do not consider this aggregation scheme in this paper.

4) Weighted-Sum Aggregation: In a generic problem setting, different scenarios may have different importance. For example, if p-th scenario happens more often than q-th scenario, then a large weight can be used for the p-th scenario and the following weighted-sum function can be used:

$$\sqcup_{k=1}^{K} f(k,\mathbf{x}) = \frac{1}{\sum_{k=1}^{K} w_k}\sum_{k=1}^{K} w_k f(k,\mathbf{x}), \tag{5}$$

Certainly, other aggregate methods are possible, but the above presents the most common strategies. We now discuss aggregate strategies for handling constraints under multiple scenarios.

B. Handling Constraint Functions

A constraint function determines whether a solution is feasible or not. For example, if the stress developed in a structure must be at most the strength of the material chosen for building the component, this constraint must be computed and satisfied for every loading condition (say k-th scenario). A check on the average of constraint values, weighted-sum of constraint values or other methods discussed above is not enough to say that the solution is feasible – the solution must satisfy every constraint for every scenario. That is, only the worst-case aggregate of the constraint function is applicable for handling constraints under multiple scenarios. Thus, since the original constraint is $g_j \geq 0$, we always adopt the following aggregation function for handling constraints:

$$\sqcup_{k=1}^{K} g_j(k,\mathbf{x}) = \min_{k=1}^{K} g_j(k,\mathbf{x}) \geq 0. \tag{6}$$

III. Proposed Multi-objective Multi-Scenario Approach

The above approaches do not change the dimension of the optimization problem. That is, the objective function and constraint functions are modified by certain transformations and the original single-objective optimization problem remains as a single-objective problem. To solve such problems, a single-objective optimization approach (a classical method [5][6] or an evolutionary approach [7]) can be used to find the optimal solution.

However, the multi-scenario single-objective optimization problem (given in Equation 1) can be converted into a multi-objective optimization problem, as follows:

$$\begin{array}{rl} \text{Minimize} & (f(1,\mathbf{x}), f(2,\mathbf{x}), \ldots, f(K,\mathbf{x})), \\ \text{subject to} & \sqcup_{k=1}^{K} g_j(k,\mathbf{x}) \geq 0, \quad j = 1,2,\ldots,J, \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)}, \quad i = 1,2,\ldots,n. \end{array} \tag{7}$$

That is, the objective function computed for each scenario now can be considered as a separate objective function, thereby posing a multi-objective optimization problem. Although the original problem is not a multi-objective one, the multi-objectivization of the problem may help solve the problem better, as found in other contexts [15][16]. The constraint function can be considered using Equation 6. Since each objective for each scenario is minimized, the above problem may not have a single minimal solution, particularly if scenarios are so different from each other that they constitute a conflicting situation among multi-scenario objective values [8]. For two or three scenario problems, elitist non-dominated sorting genetic algorithms (NSGA-II) [9] can be used, whereas for more than three scenario problems, recently proposed NSGA-III [10] can be used. Any trade-off Pareto-optimal solution will still satisfy all constraints and under all K specified scenarios. Under conflicting situation, a single trade-off solution must then be chosen for implementation. The information of weights for different scenarios can be used to perform a post-optimality decision making to choose a preferred solution. The MCDM literature [11][12] and recent EMO-MCDM studies [13][14] suggest a number of methodologies for this purpose. The knowledge of multiple trade-off solutions also allow a user to have a better knowledge of different alternative solutions which may be helpful in a longer run.

However, most real-world problems are multi-modal, meaning that every scenario may give rise to a number of optimal solutions. Although there may exist a common optimal solution for all K scenarios, finding that solution may be a difficult task. Figure 2 illustrates this aspect for a hypothetical two-scenario maximization problem. Solutions A to B are optimal for Scenario 1 and solutions A to C are optimal for Scenario 2, but the only solution A is common to both scenarios. Thus, when the problem is solved with Scenario 1 alone, any point within AB can be obtained (as they maximize the objective under Scenario 1). But any solution on AB other than A does not simultaneously maximize the objective function under Scenario 2. Posing the two-scenario problem as a two-objective problem and solving for non-weak Pareto-optimal solutions makes solution A as the target. NSGA-II [9] or NSGA-III [10] approaches can be used for this purpose. Although a single

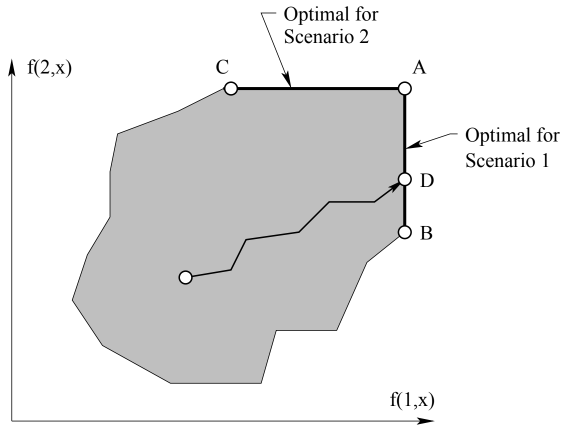

Fig. 2. Multiple scenarios are treated as a multi-objective optimization problem. All feasible solutions are represented by the shaded region. Solution A maximizes both scenarios.

solution is our target, due to the nature of the scenarios themselves, the problem may be relatively easier to optimize under some scenarios compared to others. In such problems, although there will be only one Pareto-optimal solution for a multi-objective optimization algorithm to find, seeking for that single Pareto-optimal solution using the aggregate approaches discussed in the previous subsection (worst-case, average, or weighted-sum or mean-variance approach) may overly emphasize on solving easier scenarios first by ignoring the more difficult scenarios in the beginning of a simulation run. In the context of the above figure, this may cause an algorithm to converge to a solution like D. When this happens, the search may have led to such a part in the variable space from where it may be difficult to come out and converge to the optimal region (AC) for the difficult scenarios. What we would like to achieve in such situations is a procedure that provides equal importance to all scenarios from the start to the finish of an optimization run without solving the scenarios in any hierarchical manner. Our multi-objectivization approach allows a holistic optimization task to be achieved, as the process would provide importance to all scenarios simultaneously and importantly would keep a useful diversity in solutions so as to not get 'stuck' dictated by any particular influential scenario.

In the next section, we discuss a multi-scenario Byzantine agreement problem. In the subsequent section, we consider three different scenarios of the same problem and demonstrate the usefulness of the multi-objective optimization approach.

IV. Byzantine Agreement Problem

As a case study we focus on synthesizing fault-tolerant programs for the Byzantine agreement problem [17][18]. We utilize an evolutionary approach to synthesize programs. In the Byzantine agreement problem, n processors communicate with each other in order to reach an agreement on a specific binary value. The problem consists of one general, g and three non-generals, j, k, and l. First, the general makes a decision and communicates it to the non-generals. After communication with the general and with each other, the non-generals need to finalize their decisions. This communication could be subject to a Byzantine fault where the Byzantine participant (general or non-general) sends incorrect values to other participants. It is necessary that eventually all non-Byzantine non-generals finalize a decision such that they satisfy a *validity* measure (if the general is non-Byzantine, then all non-Byzantine non-generals finalize their value to be equal to the decision of the general) and an *agreement* measure (if the general is Byzantine, then the finalized values by all non-generals are the same). Given the structure of the problem, each process maintains a decision variable d. We concatenate the name of the process to denote the variable of that process. Hence, $d.g$ denotes the decision of process g, $d.j$ denotes the decision of process j and so on. The decision of a non-general can be 1, 0, or $\perp$ (where $\perp$ denotes that the process has not yet received the decision from the general). Additionally, each process maintains a read-only variable b. Thus, $b.j$ denotes that process j is Byzantine. Finally, all non-generals maintain a variable f; $f.j$ denotes j has finalized its decision or whether the decision of j is temporary.

A. Program Representation

In this work, the generated programs are round-based distributed programs. The distributed system consists of a finite number of processes. Each process is associated with a set of variables, each with a finite domain. Some of these variables are read-only and modified only by faults. Other variables can be read or written by the process. These processes execute in a round-based manner, i.e., in round r, each process receives a message from (zero or more) other processes. It utilizes the information received in these messages to update its own state. Thus, the program of each process has the structure shown in Figure 3.

```
Actions for variable $x$
    if        condition 1    then $x = x_i$
    elseif    condition 2    then $x = 0$
    elseif    condition 3    then $x = 1$
Actions for variable $y$
    if        condition 1    then $y = 0$
    elseif    condition 2    then $y = 1$
              ...            ...
```

Fig. 3. Program structure for each process is given above

The conditions shown in Figure 3 are Boolean expressions involving the state of the process and the messages received by the process in the previous round. Statements update the variables of the process and send messages that will be used in the next round. For brevity of modeling, we partition the variables of the process into public and private variables. Intuitively, in each round, the process sends its updated public variables.

The conditions of a statement are evolved by genetic programming (GP), but the statements themselves are not. Instead, they are specified by the designer. The approach to design these statements is to choose a writable variable of the given process and assign a value to it from its domain. For example, if x is a Boolean variable of the program, then two possible statements are $x = true$ and $x = false$. Other actions could be chosen by considering the specification, e.g., if a variable is used to identify a distance from a fixed node, it could be incremented or decremented based on messages received by the process. We choose all variables that a process could read and write. Then, for each variable, we construct actions by assigning to that variable all possible values allowed by the program specifications. For a writable variable, the process in each round runs one of the actions for this variable, or none of them, based on the conditional statements evolved by GP.

B. Genetic Programming

The conditions shown in Figure 3 are evolved using a stack-based GP [19][20]. The Stack-based GP has several advantages in producing round-based programs. It is able to directly perform on the linear statements, guarantees the safety of resulting programs [21], and is not effected by noneffective code (introns). The conditions shown in Figure 3 regulate the execution of actions that are the targets of evolution. Conditional statements consist of a series of Boolean conditions connected by Boolean operators ($\wedge$ or $\vee$). Each Boolean condition consists of any one of the variables a process could read and compare with any value from its domain or the same type of variable. Genotype used in stack-based GP is comprised of one or more genomes that are represented as a vector of integers. Each genome that represents a series of conditional statements for a process is decoded into conditions and Boolean operators, and these conditions and operators are pushed into the operand stack and operator stack respectively to obtain the corresponding program. A generated program consists of multiple series of such statements.

V. Simulation Results

All simulations use a population size of 100 and the genome size is 80. A single-point crossover operator with $p_c = 0.95$ and an integer-wise mutation with $p_m = 0.0375$ are used. We set the maximum number of generations to 500. All experiments are run on an intel CORE-i7 (2.9 GHz) machine with 8 GB RAM. Each solution is tested in three different scenarios, and for each scenario, we run the generated program for a pre-specified number of times checking how many properties the program satisfies.

A. Modeling Objective Functions

The performance measure of a solution to the Byzantine agreement problem must be considered for three different criteria:

- **Validity:** If the general is non-Byzantine, then the final decision of a non-Byzantine non-general must be the same as that of the general.
- **Agreement:** The final decision of two non-Byzantine processes must be the same.
- **Termination:** All non-Byzantine non-generals must eventually finalize their decisions.

Using the above criteria, we evaluate a solution for three different scenarios: (1) when no process is Byzantine, (2) when some non-general is Byzantine, and (3) when the general is Byzantine. The goal of the GP is to evolve programs that work well in all these scenarios. Next, we describe how the objective function is constructed for each scenario.

Scenario 1: No Byzantine process. The objective value of individuals is calculated as the average of two distinct cases: one in which the decision of the general is 0, and another in which it is 1. For each of these cases, the objective function of the individual is based on the nine properties that have to be satisfied when there is no Byzantine process. Hence, for the objective for Scenario 1 is defined as follows:

From validity: $Q_1 = (d.j == d.g), Q_2 = (d.k == d.g), Q_3 = (d.l == d.g)$

From agreement: $Q_4 = (d.j == d.k), Q_5 = (d.k == d.l), Q_6 = (d.j == d.l)$

From termination: $Q_7 = (f.j == 1), Q_8 = (f.k == 1), Q_9 = (f, l == 1)$

The objective function $F_{no\_byz}(P)$ is now calculated by counting the number of properties that are satisfied and normalized to one. Thus,

$$F_{no\_byz}(P) = \frac{1}{2}\sum_{d.g=1}^{2}\frac{1}{9}\sum_{i=1}^{9} f(P \models Q_i). \tag{8}$$

Scenario 2: One of the non-generals is a Byzantine. In the following discussion, let l be the Byzantine process of a non-general. In round-based computation, each process sends some information to other processes in each round. Modeling Byzantine process in this context is straightforward: a Byzantine process sends random decision values to other processes (Possibly different decision values to different processes). If l is the Byzantine process then only Q_1, Q_2, Q_4, Q_7 and Q_8 (denoted as $Q_{relevant_l}$) have to be satisfied. Hence, we evaluate the candidate program for each of these properties. This experiment is repeated for cases where the initial decision of the general is 0 or 1 as well as for the cases where processes j and k are Byzantine. A total of 120 runs are made in this scenario. Since this involves a total of six possible experiments, each of which identifies how the five properties are satisfied, the objective function is defined as follows:

$$F_{byz\_non\_general}(P) = \frac{1}{2}\sum_{d.g=1}^{2}\frac{1}{3}\sum_{byz}^{j,k,l}\frac{1}{5}(f(P \models Q_{relevant_byz}). \tag{9}$$

Scenario 3: The general is a Byzantine. If the general is a Byzantine, then only $Q_4 - Q_9$ need to be satisfied. This experiment is repeated $n = 20$ times and the objective function is the average value of how these six properties are satisfied in each experiment. The objective function is computed as follows:

$$F_{byz\_general}(P) = \frac{1}{n}\sum_{m=1}^{n}\frac{1}{6}\sum_{i=4}^{9} f(P \models Q_i). \tag{10}$$

In all three scenarios, the objective function is maximized. Each of these objective functions has a maximum value of one, and there exists at least one solution P that will correspond to the maximum of all three objectives. We are now ready to discuss results of our simulations.

B. Worst-Case Aggregation Results

In the worst-case aggregation, the objective function is computed as the worst performance among the three scenarios:

$$F(P) = \min\left(F_{no\_byz}(P), F_{byz\_non\_general}(P), F_{byz\_general}(P)\right).$$

Figure 4 shows the variation of population-average objective value of each scenario with the generation number. It is clear

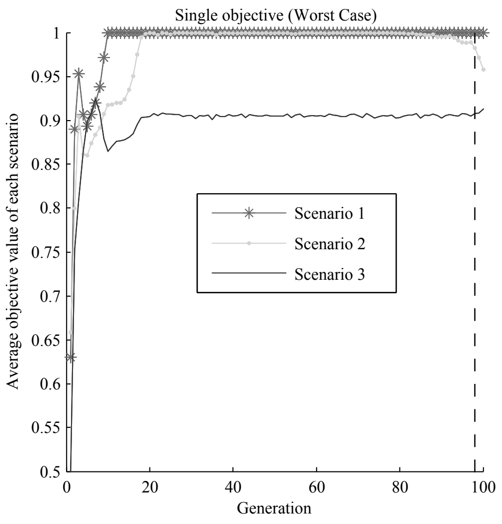

Fig. 4. Variation of three objectives with generation is shown for the worst-case aggregation method. The vertical dashed line shows the generation when the optimal solution for all three objectives (equal to one) is obtained for the first time.

from the figure that F_{no_byz} and $F_{byz\_non\_general}$ reach their maximum values quickly, whereas $F_{byz\_general}$ takes a large number of generations to come close to its maximum value. In this case, the optimal solution $P*$ that maximizes all three objectives is found at generation 98. The corresponding solution is presented in Figure 5.

```
Actions for $d.j$
if      $(d.j == \perp) \wedge (f.j \neq 1) \wedge (d.j \neq 1)$   then  $d.j = d.g$
elseif  $(d.k == d.l) \wedge (d.k == 0)$                        then  $d.j = 0$
elseif  $(d.k == d.l)$                                          then  $d.j = 1$

Actions for $f.j$
if      $(d.j \neq \perp) \wedge ((d.l \neq \perp) \vee (d.l == \perp))$
        $\wedge((d.l == d.j) \vee (d.k == d.j))$                 then  $f.j = 1$
```

Fig. 5. One of the Generated Solution for Byzantine Agreement Program

Although the optimal solution was found eventually, due to the unequal emphasis for all three objectives in the worst-case aggregation scheme, it was difficult for the GP to find the optimal solution quickly.

C. Average-Case Aggregation Results

The three objective functions are simply averaged here and the average function value is maximized. Figure 6 shows the variation of population-average value of all three objectives. A similar observation can be made here as well, however, the convergence to the true optimal solution is quick. Due to the averaging effect, although all three objectives get emphasized, the process still sets a hierarchical importance to three objectives.

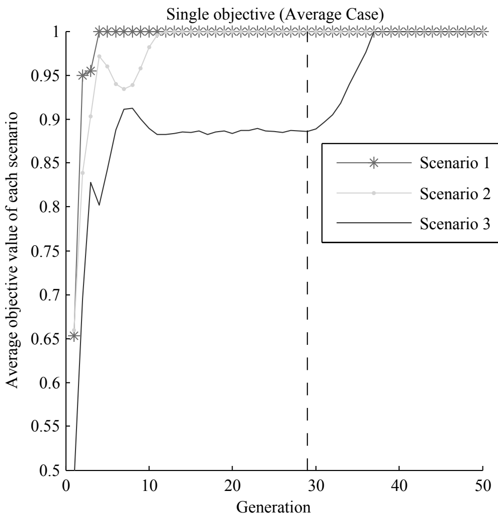

Fig. 6. Variation of three objectives with generation is shown for the average-case aggregation method.

D. Mean-Variance Aggregation Results

Next, we use the following aggregate function derived from three objective values:

$$F(P) = \mu\left(F_{no\_byz}(P), F_{byz\_non\_general}(P), F_{byz\_general}(P)\right) \\ -2\sigma\left(F_{no\_byz}(P), F_{byz\_non\_general}(P), F_{byz\_general}(P)\right).$$

and maximize $F(P)$. Figure 7 shows the variation of population-average objective values. Due to the consideration of both (increasing) mean and (reducing) standard deviation of objective values, all three objectives get emphasized and the optimization method is able to quickly converge to the true optimal solution.

E. Weighted-Sum Aggregation Results

The worst-case and average-case aggregation results have shown that it is relatively easy to solve $F_{no\_byz}(P)$ and $F_{byz\_non\_general}(P)$, and in both cases the algorithm waits until it solves the third objective to find the true optimum. If for this reason or for another reason of preferring the third objective more than the first two objectives, we use the following weighted-sum of three objectives as an aggregation scheme:

$$F(P) = 0.25F_{no\_byz}(P) + 0.25F_{byz\_non\_general}(P) \\ +0.5F_{byz\_general}(P),$$

and maximize $F(P)$. Figure 8 shows the variation of population-average objective values. Due to the emphasis put on the third objective now, the third objective reaches its maximum quickly, but the the first two objective takes a long time to converge to their maximum values. Note that for the third objective, there exist a number of optimal solutions. Unfortunately, the algorithm converges to one solution that does not correspond to the maximum of the first two objectives. It takes the algorithm 200 generations to find the optimum that is shared between all three objectives. Therefore, if different weights need to be used for different objectives, the use of a single weighted-sum of objectives ($F(P)$) may not be the right way forward.

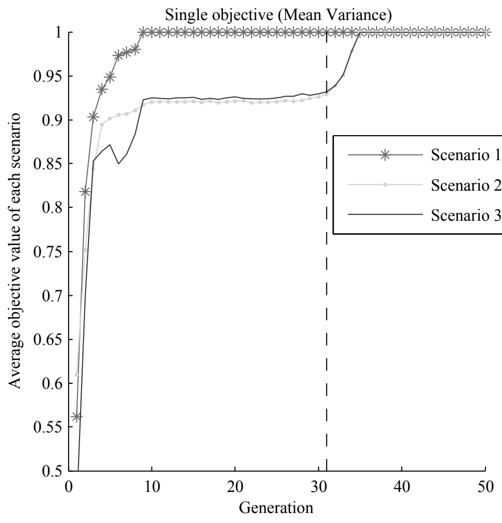

Fig. 7. Variation of three objectives with generation is shown for the mean-variance aggregation method.

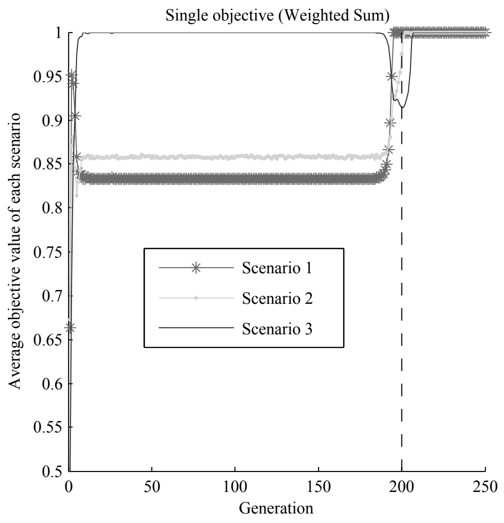

Fig. 8. Variation of three objectives with generation is shown for the weighted-sum aggregation method.

F. Multi-objective Results

We employ NSGA-II procedure and solve the Byzantine agreement problem as a three-objective optimization problem. NSGA-II does not require any additional parameters and identical genetic operators are used here. Figure 9 shows the population-average objective values with generation. A comparison of this figure with that obtained for all the previous single-objective methods reveals an interesting fact. All three objectives increase more or less in a similar manner. Since all objectives are emphasized simultaneously in a multi-objective optimization problem, the population maintains a good diversity of solutions and no objective is ignored or less-emphasized. The optimal solution for all three objectives is also obtained quickly by this method.

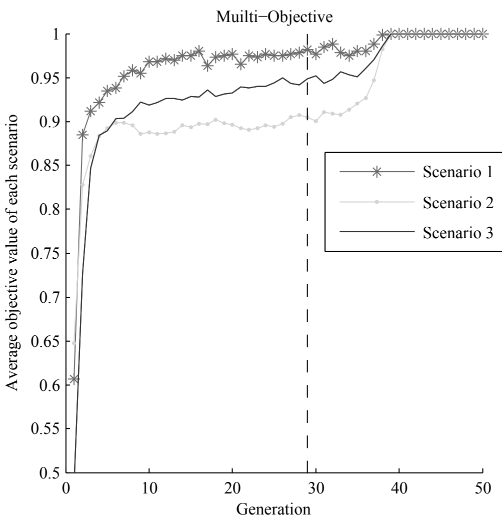

Fig. 9. Variation of three objectives with generation is shown for the multi-objective method.

G. Specific Bi-objective Aggregation Results

When the number of scenarios is large, the above multi-objective approach will be required to handle many objectives. Unfortunately, NSGA-II or other domination-based evolutionary approaches are not adequate in handling more than three or four objectives. However, recently proposed decomposition-based methods such as NSGA-III [10] or MOEA/D [22] are potential algorithms for handling many scenarios. Another approach to handling many scenarios would be to club a few scenarios together into one class and thereby reduce the number of scenarios for optimization. To demonstrate this method, we merge the first two scenarios into one and compute the combined objective function as $F_1(P) = (F_{no\_byz}(P) + F_{byz\_non\_general}(P))/2$ and use the third objective as the second objective for a bi-objective optimization ($F_2(P) = F_{byz\_general}(P)$). NSGA-II is then employed to solve this specific bi-objective problem.

Figure 10 shows the variation of the population-average objective value of each of the three original objectives. The growth of three objective values is closer to each other, meaning that the optimization algorithm is able to emphasize all three objectives in a similar manner when arriving at the optimal solution.

To summarize the outcome of all six methods, in Table I, we tabulate the number of successful runs and best, median and worst number of generations (of 25 runs) required to

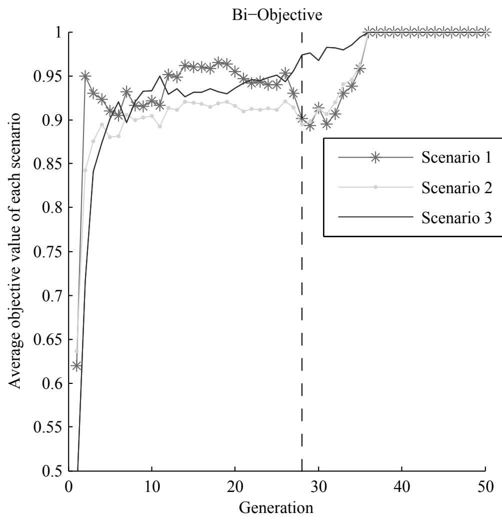

Fig. 10. Variation of three objectives with generation is shown for the bi-objective method.

arrive at the optimal solution $P*$ that maximizes all three objectives. The best values are shown in bold font. The

TABLE I. PERFORMANCE OF SIX ALGORITHMS FOR SOLVING BYZANTINE AGREEMENT PROBLEM.

Optimization Approach	Success Rate (%)	Min # Gens. to Converge	Median # Gens. to Converge	Max # Gens. to Converge
Worst-Case	92	8	98	172
Average-Case	92	6	29	**111**
Mean-Variance	96	6	31	366
Weighted-Sum	48	5	200	421
Multi-objective	**100**	5	29	386
Bi-objective	96	**4**	**27**	349

table shows that multi-objective methods are not only more successful, but they are also quicker in most of their runs. Among the single-objective methods, average-case and the mean-variance methods are better for this problem.

VI. DISCUSSION

It is clear that multi-objective approach is able to maintain an equal emphasis on each of the three objectives from the start to the end of the optimization process. The reason for this behavior is the natural diversity among optimal solutions for different objectives that an evolutionary multi-objective optimization (EMO) method maintains. To investigate this aspect further, we compute a diversity measure as follows.

At every generation, we compute the centroid $\bar{\mathbf{F}}$ of the population members in the objective space. Then, we compute the the distance $d_i = \sqrt{\sum_{i=1}^{3}(f_i - \bar{f}_i)^2}$ of each objective vector from the centroid. The diversity measure is then calculated by taking an average of the distance values, or $D = \sum_{i=1}^{N} d_i/N$. When this measure is plotted with generation counter, it will indicate the diversity of the population in the objective space. Figure 11 plots this diversity metric for all six methods. It is clear from the plot that single-

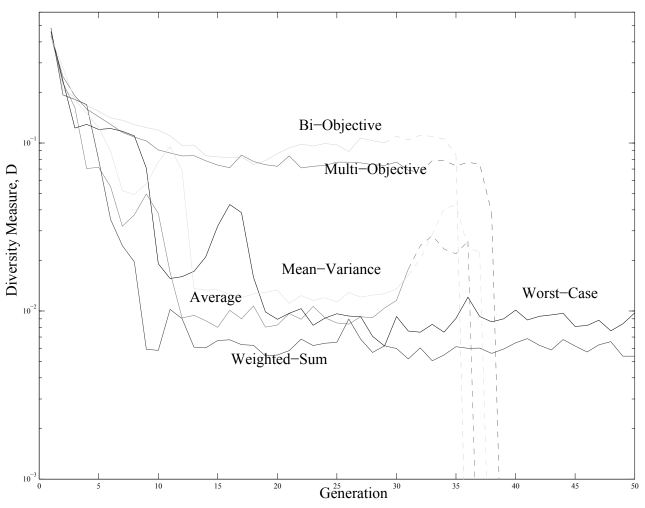

Fig. 11. Diversity measure D with generation counter for six methods of this study.

objective methods lose diversity very quickly, whereas both bi-objective and three-objective methods are able to maintain a large diversity all along. For a method, if the optimal solution $P*$ is found, the variation after its occurrence in the population is marked with a dashed line. A reason for poor performance of some of the single-objective methods is their rapid loss of diversity. Once the population gets stuck to the optimum of one of the objectives and the population diversity is small, evolutionary algorithms have difficulties in getting out from there. In multi-scenario problems, an optimal solution for all scenarios is the target, and getting stuck to an optimum for one scenario is often detrimental in arriving at the desired solution. It is now amply clear that due to presence of diversity in the population throughout the entire evolution process, multi-objective methods have performed well in the Byzantine agreement problem.

VII. Conclusions

In this paper, we have presented different methods for addressing multi-scenario optimization problems which often occur in practice. We then have discussed various aggregation methods of deriving a single objective function and constraint function value, which then can be used in an optimization algorithm. We have also proposed generic and specific multi-objective optimization approaches in which objectives for different clusters of scenarios can be optimized as a multi-objective manner. On a Byzantine agreement problem, we have demonstrated that the multi-objective approaches are able to maintain adequate diversity of the solutions so generation operators of an evolutionary algorithm are able to emphasize each objective function arising from every scenario uniformly to eventually find the desired optimum without getting stuck anywhere in the search space.

The idea portrayed and demonstrated in this paper is generic and is ready to be applied to other multi-scenario optimization problems from engineering and other computational optimization fields.

References

[1] C. D. Laird and L. T. Biegler, *Large-Scale Nonlinear Programming for Multi-scenario Optimization*. Springer, 2008, pp. 323–336.

[2] D. Varvarezos, L. Biegler, and I. Grossmann, “Multi-period design optimization with sqp decomposition,” *Comp. Chem. Eng.*, vol. 18, no. 7, pp. 579–595, 1994.

[3] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira, “HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance,” in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, 2006.

[4] P. -L. Aublin, S. B. Mokhtar, and V. Quéma, “RBFT: Redundant Byzantine Fault Tolerance,” in *33rd IEEE International Conference on Distributed Computing Systems*, 2013.

[5] K. Deb, *Optimization for Engineering Design: Algorithms and Examples*. New Delhi: Prentice-Hall, 1995.

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

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

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

[9] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, “A fast and elitist multi-objective genetic algorithm: NSGA-II,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[10] K. Deb and H. Jain, “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*, in press.

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

[12] V. Chankong and Y. Y. Haimes, *Multiobjective Decision Making Theory and Methodology*. New York: North-Holland, 1983.

[13] J. Branke, K. Deb, K. Miettinen, and R. Slowinski, *Multiobjective optimization: Interactive and evolutionary approaches*. Berlin, Germany: Springer-Verlag, 2008.

[14] K. Deb, J. Sundar, N. Uday, and S. Chaudhuri, “Reference point based multi-objective optimization using evolutionary algorithms,” *International Journal of Computational Intelligence Research (IJCIR)*, vol. 2, no. 6, pp. 273–286, 2006.

[15] J. D. Knowles, R. A. Watson, and D. W. Corne, “Reducing local optima in single-objective problems by multi-objectivization,” in *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO-01)*, 2001, pp. 269–283.

[16] D. Saxena and K. Deb, “Trading on infeasibility by exploiting constraint’s criticality through multi-objectivization: A system design perspective,” in *Proceedings of the Congress on Evolutionary Computation (CEC-2007)*, in press.

[17] L. Lamport, R. Shostak, and M. Pease, “The byzantine generals problem,” *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, Jul. 1982. [Online]. Available: http://doi.acm.org/10.1145/357172.357176

[18] S. S.Kulkarni, A. Arora and A. Chippada, “Polynomial time synthesis of Byzantine agreement,” in *IEEE Symposium on Reliable Distributed Systems(SRDS)*, 2001, pp.130–140.

[19] L. Spector and A. Robinson, “Genetic programming and autoconstructive evolution with the push programming language,” in *Genetic Programming and Evolvable Machines*, 2002, pp. 7–40.

[20] T. Perkis, “Stack-based genetic programming,” in *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*. IEEE, 1994, pp. 148–153.

[21] M. Oltean, C. Grosan, L. Diosan, and C. Mihaila, “Genetic programming with linear representation: a survey,” *International Journal on Artificial Intelligence Tools*, vol. 18, no. 2, pp. 197–238, 2009.

[22] Q. Zhang and H. Li, “MOEA/D: A multiobjective evolutionary algorithm based on decomposition,” *Evolutionary Computation, IEEE Transactions on*, vol. 11, no. 6, pp. 712–731, 2007.