Robust Optimization with Multiple Solutions

Peng Yang
Supervisors: Prof. Xin Yao and Prof. Ke Tang

***Abstract*—Variations are pervasive in the real-world optimization problems. Among them, robustness against environmental variation has been extensively studied. It concerns about the changes in the environmental parameters, and is the focus of this paper. As the environmental parameters change, there are actually multiple static environments linked up. When optimizing for multiple environments, one usually needs to sacrifice performance in one environment in order to gain better performance in another. However, there may not be a single solution that meets the performance requirements for all environments. In this paper, we propose to find multiple solutions that each serve a certain group of environments. This formulation is named as Robust Optimization with Multiple Solutions (ROMS) in this paper. Based on two basic observations of ROMS, two evolutionary algorithms based approaches to ROMS are proposed, namely direct evolution and two-phase evolution. A benchmark problem generator is also suggested to produce uniform-random ROMS problems. The two approaches are then empirically studied on a variety of synthetic problems.**

***Index Terms*—Robust optimization, Multiple solutions, Evolutionary algorithms and Environmental variation.**

I. Introduction

Variations are pervasive in the real world. Neglecting them in optimization may result in optimal solutions that are practically unstable and ultimately useless. A case study reported by [1] found that in 13 out of 90 linear programs drawn from Netlib, a small 0.01% relative perturbation applied to the program coefficients can result in serious constraint violation by more than 50% at the optimal solution. Using such optimal solutions in practice can sometimes be quite risky. A robust solution, on the other hand, is one that performs reasonably well even when subject to variations. It is thus often necessary to explicitly incorporate variations into the optimization model and look for robust solutions. Such practice is called robust optimization by [2].

Many variations may be taken into account when referring to robust optimization [2]–[4]. Among them, robustness against environmental variation has been extensively studied [5]–[9]. It concerns about the changes in the environmental parameters, and is the focus of this paper. When the environmental parameters change, the landscape of the problem changes. Therefore, if the environmental parameters of a problem change several times, there are actually multiple corresponding landscapes, or say environments, to be solved. The quality of a solution normally changes, subject to the environmental variation. A robust solution, in this scenario, is thus defined as a solution that exhibits reasonable systemic performances over all environments.

To measure the overall quality of a solution, two formulations that output scalar values have been widely used. The first formulation, called the average-case analysis, is defined as the average of the objective values of a solution over the environments; while the second one is defined as the worst objective value, which is therefore called the worst-case analysis. In the literature, various approaches have been proposed for these two formulations, e.g., Monte Carlo methods [6], [7], [10], [11] for average-case analysis and competitive co-evolutionary methods [9] for worst-case analysis. It is also suggested to formulate the robust optimization problem as a Multi-Objective Optimization (MOP) problem by considering more than one overall objective function simultaneously [12]. In such cases, variance of the objective values over the environments is used as an additional (overall) objective by Jin and Sendhoff [13]. One may also encounter the situation that the constraints will vary corresponding to the environmental variations. When dealing with hard constraints, it is reasonable to require them being satisfied in all environments. When the environment is modelled as a random variable following some distribution, a more general choice is to formulate the constraints as probabilistic inequalities called chance constraints [14]. The practice of formulating and solving chance-constrained problems is called reliability-based optimization in the literature [8], [15], [16].

The above-mentioned robust formulations all aim to find a single solution. Although MOP-based formulation can output a set of non-dominated solutions, one still needs to select the preferred one manually. In practice, however, there may not exist a single solution that meets the requirements over all environments. This may be due to either constraint violation or poor objective values. The former case happens when the feasible regions in different environments are disjoint, and a single solution feasible in all environments cannot be found. The reason for the latter case lies in what is named as the price of robustness by Bertsimas and Sim [17]. More often than not, a solution must sacrifice certain performance (in terms of the objective function) in one environment in order to gain some performance boost in another. With this compromise, there may not exist a tradeoff solution that meets the performance requirements in all environments. A possible way out of this dilemma is to resort to multiple solutions. Concerning constraints, we may find a solution to serve each group of environments that share a common feasible region. Concerning the performance, each solution could serve a group of similar environments. With a larger number of solutions, the variety of environments that a solution needs to serve is reduced and thus the cost of robustness lowered. Consequently, less compromised performance in the environments can usually be achieved. The number of solutions marks the tradeoff preference between robustness and performance, and is assumed to be specified by the user. Ultimately, we are faced with the problem of finding a set of solutions of a given size for the environments. We call this generalized formulation robust optimization with multiple solutions (ROMS)[1].

To solve ROMS, Evolutionary Algorithms (EAs) are preferred since their population-based nature well fits the tasks of finding multiple solutions in a single run. Many examples can be seen in Evolutionary Multi-objective Optimization [18], Evolutionary Multi-modal Optimization [19] and so on. Specifically, two EAs-based approaches to ROMS are proposed in this paper. The first approach, called direct evolution, reduces the problem from a mixed-integer optimization problem to a simpler real-value optimization problem which is then solved by a direct application of EAs. The second approach consists of two-phase, which forms a circle by looping two optimization procedures iteratively. To compare the two algorithms, a benchmark problem generator is suggested to generate uniform-random ROMS problems based on a given basis function. Using this generator, the algorithms are then empirically studied on a variety of synthetic problems.

The main contributions of this paper include:

1. Formally defining the ROMS problem, and suggesting a benchmark function generator.

2. Demonstrating how ROMS could be addressed with an existing EA directly. Besides, a novel approach that is more specialized for ROMS is also proposed.

The rest of this paper is organized as follows. In the next section, formal definition of ROMS is given. A discussion of its properties follows in Section 3. Based on the properties, two evolutionary approaches are proposed in Section 4 and 5, respectively. A benchmark problem generator is suggested in Section 6, followed by the empirical studies. Finally, some conclusions are drawn in the last section.

II. Formal Definition

Without loss of generality, we consider the unconstrained scenarios in this paper. An unconstrained single-objective optimization problem can be described as follow,

$$\text{optimize} \quad f(x), x \in \chi \tag{1}$$

where f is the objective function, and χ denotes the search space. When environmental changes are taken into account, the objective function is extended by introducing the environmental parameter α, as follow,

$$f(x, \alpha), x \in \chi, \alpha \in C \tag{2}$$

where C denotes possible settings of the environment [3]. When considering fluctuations around one nominal environment, C is a connected continuous region. Here, we consider m nominal environments, and C should be a discrete set $C = \alpha_1, ..., \alpha_m$. For convenience, we denote $(\cdot, \alpha_i)$ by f_i,

[1]Part of this work has been presented in the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Nov., 2014, Singapore.

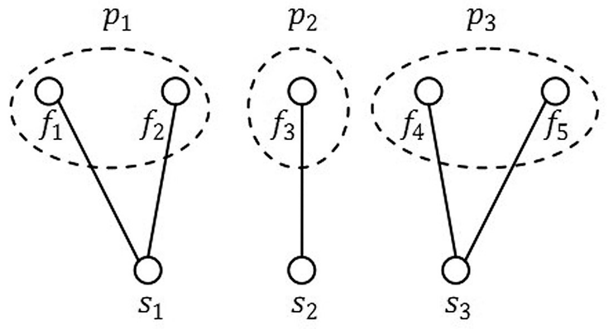

Fig. 1: This bipartite graph illustrates an example of ROMS when $m = 5$, $n = 3$.

and call it an environment. The m environments constitute a function set $FS = f_1, , f_m$. Given an x, its overall objective value is evaluated by some function $\widehat{f}$ based its individual objective values in each environment. The robust formulation for finding a single solution x then can be formulated as:

$$\text{optimize} \quad \widehat{f}(f_1(x), ..., f_m(x)), x \in \chi \tag{3}$$

Two widely used choices for $\widehat{f}$ are the average function:

$$\text{optimize} \quad f_{avg}(f_1(x), ..., f_m(x)) = \frac{1}{m}\sum_{i=1}^{m} f_i(x) \tag{4}$$

and the one that returns the worst objective value:

$$\text{optimize} \quad f_{wst}(f_1(x), ..., f_m(x)) = \text{min/max}(f_1(x), ..., f_m(x)) \tag{5}$$

The specific meaning of "worst" depends on whether the problem is a maximization or minimization problem, in which cases, the functions min and max are used, respectively. As can be seen, in this traditional robust formulation, the single solution x needs to balance the objective values among all m environments, and a severe performance compromise is expected.

ROMS can be formulated by extending the robust formulation to the case of multiple solutions. Suppose the desired number of solutions is n. That is, a set of n solutions $S = s_1, ..., s_n$ is used to cope with all the environments in FS, with each single solution serves a subset of FS. The concept is best illustrated using a bipartite graph. Fig. 1 shows an example when $m = 5$, $n = 3$. A proper bipartite graph for ROMS should satisfy the property that each vertex representing an environment has exactly one associated edge. Intuitively, this guarantees that each environment is served by exactly one solution. From Fig. 1, it can be seen that not only the solution set S but also the service relationships between the solutions and environments must be determined. The relationships can be specified by a mapping M from FS to S such that $(f_i) = s_j$ if and only if environment f_i is served by solution s_j. A complete solution to ROMS is thus a pair (S, M), and the problem can be described as:

$$\text{optimize} \quad \widehat{f}(S, M) = \widehat{f}(f_1(M(f_1)), ..., f_m(M(f_m))) \quad (6)$$

where $\widehat{f}$ is the overall objective function as in the traditional robust formulation.

In this work, both f_{avg} and f_{wst} for the overall objective function are considered. However, for convenience, the following discussions will only refer to f_{avg}. At the end of Section 5, it is shown that the proposed approaches also apply to f_{wst} without any difficulty.

III. Basic Observations

It is easy to see that there are two components S, M to be optimized, which are of different types. The first observation we made is that given a solution set S, its best accompanying M can be computed exactly in polynomial time.

Observation 1: Given a solution set S, the optimal accompanying M is one that (assuming a minimization problem):

$$\forall f_i : [\forall s_j : f_i(s_j) \geq f_i(M(f_i))] \quad (7)$$

In other words, the optimal M is one that maps each objective function to the solution in S that performs best on it. The algorithm to compute such an M is trivial and costs $O(mn)$ objective function evaluations.

To see the second observation, a partition P of FS is first introduced. By definition, all conditions served by a common solution are required to be in a single part. A P is therefore can be derived from an M by grouping together all conditions that are mapped to the same solution. Conversely, each P can also be viewed as defining an M with the specific solutions unspecified (represented by placeholders).

Observation 2: Given a partition $P = p_1, , p_k(kn)$ of FS, in which the objective functions are required to be mapped to a single solution, the optimal solution set S is one that

$$\forall p_i : [\exists s_j : s_j \text{ optimizes } p_i] \quad (8)$$

where

$$s_j \text{ optimizes } p_i \Leftrightarrow s_j \text{ minimizes } \frac{1}{|p_i|} \sum_{f_q \in p_i} f_q \quad (9)$$

which amounts to an ordinary optimization problem. This enlightens us to compute an optimal S from a given P. The procedure is to solve for each part p separately via ordinary optimization and put together the obtained solutions to get S. since P can be viewed as defining an M (strictly speaking, an M pattern), this second observation can be seen as a reversal to the first one, i.e., to compute an optimal accompanying S from a given M. In the following, according to the context, M may be used to mean its defining P. In the next two sections, two EAs-based approaches to solve ROMS are inspired by the basic observations.

Algorithm 1: ROMS-DE

```
initialize population pop
evaluate pop
for each new generation
    for i ← 1 to population size ps = |pop|
        randomly select a, b, c ∈ [1..ps] that are mutually different and also different from i
        diff ← pop[a] + F × (pop[b] − pop[c])
        randomly select r ∈ [1..n]
        for j ← 1 to n
            generate a random number rnd ∈ [0, 1]
            if rnd < CR or j = r then trial[i][j] ← diff[j]
            else trial[i][j] ← pop[i][j]        trial[i][j] is the j-th solution of trial[i]
    evaluate population trial
    for i ← 1 to ps
        pop[i] ← the better of pop[i] and trial[i]
```

Fig. 2: The pseudo code of Direct Evolution.

IV. Direct Evolution Approach

A straightforward approach is to fit the problem into the general framework of an EA and solve it directly by evolution. Originally, the individual encoding has to account for both S and M, leaving us a difficult mixed-integer optimization problem to deal with. However, thanks to Observation 1, the problem can be made easier. Concretely, since the optimal M can be computed exactly from a given S in polynomial time, it is not necessary to let the evolution process figure out the right matching between S and M. The idea is for the individuals to encode S only. When evaluating an S, its optimal M is firstly computed on the fly and then the resultant pair ¡S, M¿ is evaluated. In this way, the problem is reduced to a real-value optimization problem rather than a mixed-integer one. Compared with other real-value optimization problems, ours has a somewhat unconventional objective function whose evaluation involves computing the corresponding M. A specific instantiation of this idea comes next.

Suppose the search space for each single solution in S is R^D. A solution set S containing n solutions is directly encoded as a real vector of size $D \times n$, where each consecutive D values encode a single solution. Any real-valued EA can be employed to address this problem, and differential evolution (specifically, the standard $DE/rand/1/bin$) proposed by [20] is adopted in this work to demonstrate the proposed approach and to conduct empirical studies. This particular instantiation is denoted as ROMS-DE (short for Direct Evolution) in this paper. The pseudo-code of ROMS-DE is shown in Fig. 2 (F and CR are parameters of differential evolution).

In theoretical study of evolutionary algorithms, computational cost is measured in terms of fitness evaluation counts, assuming each takes unit cost. Given a certain fitness evaluation budget, we usually need to estimate the maximum number of generations to evolve. We now derive the fitness evaluation cost for ROMS-DE to evolve G generations, which is denoted by $cost_{DE}(G)$. Along this way, it is also shown how fitness evaluations can be saved via a carefully-made implementation and that the algorithm is less expensive than it seems from the pseudocode. First, it is important to note that a unit-cost elementary fitness evaluation here refers to the evaluation of a single solution in a single condition, rather than that of a whole solution set S on all considered conditions FS. With a straightforward implementation, it is easy to get $cost_{DE}(G) = (n \times m) \times ps \times G$. To evaluate an S, the fitness of each single solution in each condition needs to be known. That is where the term $(n \times m)$ comes from. To evaluate a whole generation thus costs $(n \times m) \times ps$ fitness evaluations. What follows is a total of $(n \times m) \times ps \times G$ fitness evaluations for evolving G generations. However, with a better implementation, the cost can be lowered. The point is that in a newly generated S ($trial[i]$ in the pseudo-code), some single solutions are directly copied from its parent $pop[i]$ with their fitness in each condition already computed. There is no need to re-evaluate them. As a result, to evaluate the newly generated S, only the solutions coming from the difference vector $diff$ needs to be evaluated in each condition. This improvement does not affect the initial generation which still costs us $(n \times m) \times ps$ fitness evaluations. For each generation that follows, we first need to estimate the number of solutions coming from $diff$ in each newly generated S. The compound condition test at Line 10 in Fig. 2 says that the $r-th$ solution always comes from $diff$ while the rest come from $diff$ with a probability of CR. Overall, for each newly generated S, the expected number of solutions coming from $diff$, which need to be evaluated in each condition is $[1 + (n-1) \times CR]$. With this improved implementation, the expected cost for evolving G generations is now reduced to

$$cost_{DE}(G) = (n \times m) \times ps + [(1 + (n-1) \times CR) \times m] \times ps \times (G-1). \quad (10)$$

V. Two-Phase Approach

The direct evolution only takes the advantage of the $Observation 1$. Although it is quite straightforward and easy to implement, it does not exploit the full benefits of the observations made above. Furthermore, one may also be interested in how the $Observation 2$ can be adopted to help design a new approach. For these two purposes, the second approach, based on the interaction between the two observations, is proposed. The first observation essentially tells a way to derive an optimal M from a given S, denoted as $S \Rightarrow M$. Likewise, the second observation essentially tells a way to derive an optimal S from a given P or an M pattern. Similarly, it is denoted as $M \Rightarrow S$. By connecting them, a loop that goes from S to M and back can be formed. Since the output of both procedures are optimal, by iterating this loop, one is actually undergoing an optimization progress. The second approach is based exactly on this idea. We call it two-phase evolution, for the optimization loop consists of two phases, namely $S \Rightarrow M$ and $M \Rightarrow S$.

A straightforward implementation of this approach goes as follows. Staring from a randomly generated S, repeat the two procedures $S \Rightarrow M$ and $M \Rightarrow S$ until some termination criterion is met. Again, as an example, $DE/rand/1/bin$ is employed to perform the optimization involved in $M \Rightarrow S$.

Algorithm 2: ROMS-TP

```
1   initialize solution pool $PL$
2   $archive \leftarrow null$
3   $M_{last} \leftarrow null$
4   for each new iteration
5       $pop_S \leftarrow$ evolve on $PL$    $\boxed{pop_S \text{ is a population of solution sets}}$
6       $archive \leftarrow better(archive$, best individual in $pop_S)$
7       $pop_S' \leftarrow \{S \in pop_S | S$ has a different corresponding $M$ from $M_{last}\}$
8       if $pop_S' \neq \emptyset$ then $S \leftarrow$ best individual in $pop_S'$
9       else $S \leftarrow$ best individual in $pop_S$
10      $M_{last} \leftarrow$ the optimal $M$ corresponding to $S$
11      $P \leftarrow$ the partition defined by $M_{last}$
12      $pops \leftarrow \emptyset$
13      $S \leftarrow \emptyset$
14      for each $p \in P$
15          $pop \leftarrow$ evolve on $p$    $\boxed{pop \text{ is a population of single solutions}}$
16          $pops \leftarrow pops \cup \{pop\}$
17          $S \leftarrow S \cup \{$best individual in $pop\}$
18      $archive \leftarrow better(archive, S)$
19      $PL \leftarrow$ build on $pops$
```

Fig. 3: The pseudo code of Two-Phase Evolution.

In accordance with direct evolution, the two-phase evolution approach can be seen as maintaining and optimizing a solution set S along the loop. However, different from direct evolution, only one S is maintained by two-phase evolution. Intuitively, in each tour around the loop, the maintained S jumps from one point to another in the search space of S. The $S \Rightarrow M$ procedure is exact. Assuming no randomness in the $M \Rightarrow S$ procedure as well, the next point to sample (jump to) is determined completely by the current one. Consequently, the whole optimization process is determined once the initial point is determined. This can become quite risky when the initial point is randomly chosen.

In an attempt to reduce the risk, the initial search coverage is increased by generating more than N solutions. Originally, a single S is randomly generated, which contains N solutions. Now, a solution pool PL of a size greater than N is randomly generated. Combinatorial optimization is then performed on the generated PL to select N solutions that together make up an optimal S with respect to the given PL [21]. Denote this procedure by $PL \Rightarrow S$. Another modification concerns the procedure $M \Rightarrow S$ that involves solving K real-value optimization problems. Since a population-based evolutionary algorithm like $DE/rand/1/bin$ is used to perform the optimization, a set of solutions is obtained for each problem. By the original design, only the best solution is selected into S. With the introduction of PL, solutions other than the best one may also be used. When the goal is to generate an updated PL that has a size larger than S, more solutions will be selected. With these modifications, the original optimization loop $S \Rightarrow M \Rightarrow S$ becomes $PL \Rightarrow S \Rightarrow M \Rightarrow PL$ with increased search power to reduce risk.

Another improvement over the original design concerns incorporating an explicit mechanism to prevent the optimization process from stagnation. In order for the optimization process to continually make progress, a different M shall be reached each time around the loop. As an example, the Genetic Algorithm (GA) (Deb 2010) is employed to perform the procedure

$PL \Rightarrow S$. Since GA is population-based, a population of S is obtained in the end, each have an accompanying optimal M. By the original design, the best S is selected with its M derived. Whereas making sense in most cases, this is not the best choice when the derived M is identical to the one derived in the previous tour around the loop. When it happens, the algorithm stagnates, i.e., simply re-producing the individuals in the previous iteration. To fix this, instead of always selecting the absolutely best S, we try to choose the best S among those that produce an M different from the one in the previous iteration.

With the aforementioned improvements and with the use of soft-computing optimization algorithms, quality of the maintained S is not guaranteed to be non-decreasing along the optimization loop. As a result, an external record is set up that maintains the best S ever encountered during the optimization process. This remedy finally makes the two-phase evolution approach complete. The two-phase evolution is denoted as ROMS-TP. The pseudo-code is given in Fig. 3.

The particular implementation used in this work and its computational complexity $cost_{TP}(G)$ (here G refers to the number of iterations around the loop rather than generations) are described in the following texts. Some implementation issues will also be discussed along the way. To make things simple, the size of PL is fixed to a predefined number. Line 5 in Fig. 3 is done in two steps. In the first step, each single solution in PL is evaluated in each condition, which costs us $|PL| \times m$ fitness evaluations. In the second step, a combinatorial optimization is done on PL using a generational genetic algorithm. Thanks to the pre-processing in the first step, the evolution process in the second step costs no fitness evaluation at all. Details of the genetic algorithm used are listed in Table 1. Lines 6 13 are cost-free. A standard $DE/rand/1/bin$ is employed to perform the real-value optimization at Line 15. In total, Lines 14 17 cost $|pop| \times m \times g$ fitness evaluations where g is the number of generations to evolve at Line 15 for the differential evolution. Lines 18-19 are cost-free. Thus, for this implementation, the required computational complexity will be:

$$cost_{TP}(G) = (|PL| \times m + |pop| \times m \times g) \times G \qquad (11)$$

Care should be taken when implementing Line 19. There are a total of $|PL| \times |pop|$ single solutions contained in $pops$, out of which $|PL|$ are to be selected to make up the new PL. Recall that the size of PL is fixed. Depending on the parameter settings, $|PL| \times |pop|$ may be smaller than, equal to, or larger than $|PL|$. In the first two cases, all solutions in pops go to PL with the remaining space, if any, being filled up by randomly generated solutions. In the last case, only part of $pops$ goes to PL. A proper selection of solutions should be made. The selection could be drawn from each population in $pops$ as evenly as possible, so that each population is well represented with minimum information loss. To do so, one solution is randomly selected (without replacement) from each population in turn repeatedly until the desired number $|PL|$ is reached.

Up to now, the discussion is restricted to the robustness measure f_{avg}. In fact, the two proposed approaches also apply to the robustness measure f_{wst}, which represents a kind of worst-case analysis. First, note that the first observation holds perfectly without any change in the f_{wst} case. The second observation holds when the meaning of the phrase "s optimizes p" in (8) is adapted to

$$s_j \text{ optimizes } p_i \Leftrightarrow s_j \text{ minimizes } \max_{f_q \in p_i} f_q \qquad (12)$$

which corresponds to the new measure. Both the proposed approaches are based on the correctness of the two observations. When the observations hold, the proposed two approaches can work.

VI. Empirical Study

A. Experimental Settings

To get an actual feeling and experience with the ROMS problem and the two approaches proposed, experiments are done on synthetic functions. In this subsection, we first describe the problem settings, and then the algorithmic settings.

A ROMS problem should be investigated from three perspectives, i.e., the number of conditions m, the desired number of solutions n, and the general function form of the conditions. In the conducted experiments, m goes from 4 to 20 with a step-size 4, and n goes from 2 to 10 with a step-size 2. The first two columns in Table 2 list the m, n combinations we use. With respect to the function forms, 5 functions, i.e., F_1-F_3, F_6, F_{10}, were chosen from the standard benchmark functions for the CEC-2005 special session on real-parameter optimization [22]. These functions cover unimodal and multimodal functions with the latter further divided into basic functions, expanded functions, hybrid composition functions and pseudo-real problems. The chosen functions are all without special properties, e.g., static noise, global optimum on bounds, unbounded search space and too time-consuming. To introduce the notion of environmental parameters, we additionally parameterize each function with a translation vector and an orthogonal matrix (matrices with orthonormal column vectors). The translation vector shifts the function while the orthogonal matrix transforms the function with a combination of rotation (around the global optimum) and reflection (via some hyper-plane through the global optimum) [23]. The conditions in FS are thus generated from the same function with different translation vectors and orthogonal matrices built in. The environmental parameters are generated as uniform random variables in their corresponding spaces. The generation of uniform random translation vectors is simple. Just generate each entry uniform-randomly. To generate uniform random orthogonal matrices, we follow the classical two-step algorithm proposed by Diaconis and Shahshahani [24]. First, generate each matrix entry randomly following a standard normal distribution, and then apply the Gram-Schmidt orthogonalization. All problems are minimization problems, and are tested in both $D = 2$ and $D = 30$ dimensions.

The parameter settings for ROMS-TP are listed in Table 3. For differential evolution, we follow the parameter settings suggested by Storn and Price [20], i.e., with $F = 0.5$, $CR = 0.1$ and a population size of 5 times the problem dimension. Notice that while for the differential evolution used in ROMS-TP, the problem dimension is D, for ROMS-DE it should be $n \times D$, which equals the length of the encoding vector of a solution set. For all experiments, a fitness evaluation budget of 1e7 is imposed. The number of iterations to run is computed accordingly to $cost_{DE}(G)$ and $cost_{TP}(G)$ respectively, as shown in Table 2.

B. Experimental Results

In this subsection, the experimental results are presented. All experiments are repeated for 25 times with the average results reported. A 95% confidence interval is also reported for each result, assuming a normal distribution. The mean objective value over the different environments is used as the robustness measure. For convenience, we label each subexperiment in the form F_x-m-n-D-f_y, so that F_1-4-2-2-f_{avg} refers to the experiment with function F_1, $m = 4, n = 2, D = 2$ and the robustness measure f_{avg}. We first present and discuss the results obtained with f_{avg} and then f_{wst}, and for each measure, $D = 2$ comes before $D = 30$. The performance of each algorithm from three perspectives is analyzed from three perspectives. The first one is optimization dynamics which is demonstrated in the 2-dimensional optimization curve (also known as convergence curve in evolutionary computation) with fitness evaluation counts as x-axis and the robustness measure as y-axis . The second and third are scalability with respect to m and n respectively.

Optimization dynamics for f_{avg}, $D = 2$: When n is fixed at 2, the big picture looks similar for different m values. For brevity, only the results for $m = 12$ are shown. The results on different functions can be clustered into two groups. For the first group (F_1 F_3, F_6), we take the results on F_1 for example (Fig. 4). Pay attention that for each algorithm, there are three lines. The middle line shows the average result, while the upper and lower ones give the bounds for the 95% confidence interval. To obtain the results, for ROMS-DE, the best-of-generation objective value, at the end of each generation, is drawn. Since differential evolution implements elitism, the best-of-generation value is also the best value seen so far. For ROMS-TP, the external archive at the end of each step is drawn, resulting in two samples for each iteration. ROMS-TP's curve starts from the end of the first step, which is the start of the second step, of the first iteration. Thus, the first drop shown by the curve, if any, is due to the second step, not the first.

The curve of ROMS-DE in Fig. 4 looks typical to a traditional evolutionary algorithm. This is due to ROMS-DE is essentially an application of standard differential evolution. The curve of ROMS-TP is more interesting. An r decrease happens at the first iteration. After that, the curve descends slowly. Besides, a careful observation will reveal that decrease happens mostly in the second step of each iteration. This is seen in the alternating "drop, level-off, drop" pattern along sample points on the curve. Comparing the two curves, premature convergence in ROMS-DE can be seen in Fig. 4. A general way to counteract premature convergence is to use a larger population, but it also slows done the convergence. Fig. 5 shows the corresponding result on F_{10} (the second group). Being contrary to the previous result, starting from the first sample point, an alternating "level-off, drop, level-off" rather than "drop, level-off, drop" pattern is seen along the curve of ROMS-TP. In other words, decrease is mainly due to the first step in each iteration. The high multimodality, which is F_{10}'s most distinct feature, may be the reason behind this phenomenon. It makes the real-value optimization involved in the second step more difficult, while having no direct impact on the first step. When the optimization effect is shifted from the second step to the first, the decrease is also more gradual. Without a dramatic decrease in the second step at the first iteration, ROMS-TP's performance is greatly weakened with a curve running always above that of ROMS-DE.

Next, the results with varying n while $m = 20$ are discussed. Again, the results can be clustered into two groups. On functions in the second group (F_3, F_6, F_{10}), the picture does not change much for different n values. On F_3 and F_6, the results are similar to Fig. 4, and on F_{10} the results are similar to Fig. 5. Taking F_1 for example, Fig. 6 shows typical results on the first group (F1, F2) as n increases. First, it can be seen that the optimization process of ROMS-DE is less well developed (or less well converged) with a larger n. Recall that the population size is set to $5 \times n \times D$ which increases with n. Thus, a natural explanation for this observation is that with a larger population, an evolutionary algorithm just needs more fitness evaluations to converge. For ROMS-TP, the following trend can be observed. With an increasing n, the optimization effect gradually shifts from the second step of each iteration to the first. What follows is consistent with the previous observation made on the comparison between Figs. 4 and 5. As the first step plays a more and more important role, descent of the optimization curve becomes more gradual. And without the remarkable decrease in the second step at the first iteration, ROMS-DE catches up with ROMS-TP.

Scalability for f_{avg}, $D = 2$: Fig. 7 gives the results of scalability test with respect to m. The objective values are the final results obtained by the algorithms at a fitness evaluation count of approximately 1e7. As can be seen that, ROMS-TP has better scalability on all test functions expect F_{10} where the opposite seems to hold somehow. Fig. 8 gives the results of scalability test with respect to n. Overall, ROMS-DE has better scalability on all test functions with the exception of F_{10} where, again, the opposite seems to hold. It is also noticed that at the given fitness evaluation count, ROMS-TP generally has obtained a final result no worse than that of ROMS-DE for each m, n combination on all test functions expect F_{10} where ROMS-DE has got the upper hand. The

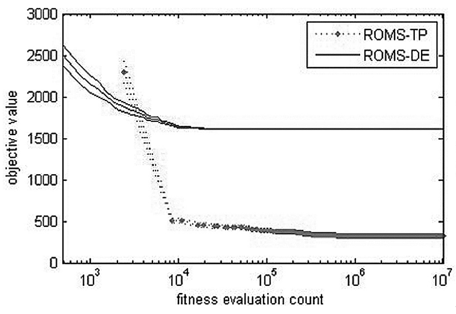

Fig4. F_1-12-2-2-f_{avg}

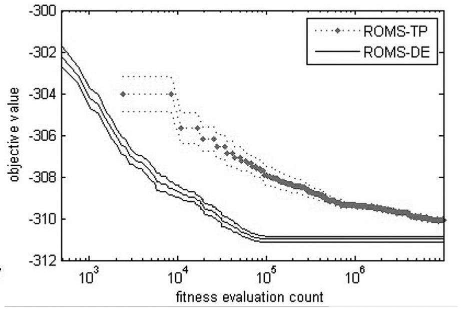

Fig5. F_{10}-12-2-2-f_{avg}

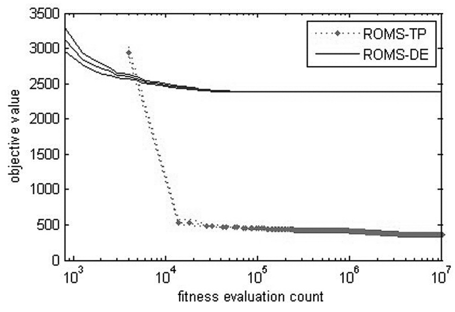

Fig6(a). F_1-20-2-2-f_{avg}

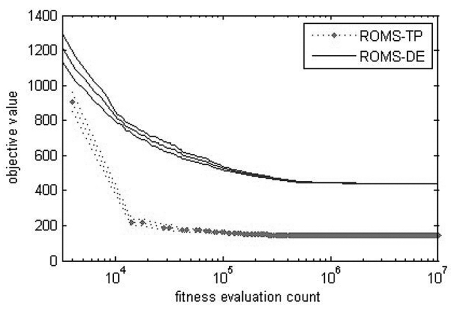

Fig6(b). F_1-20-4-2-f_{avg}

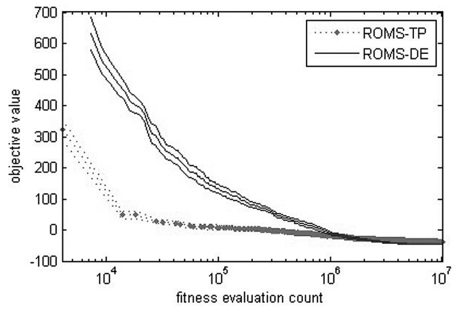

Fig6(c). F_1-20-6-2-f_{avg}

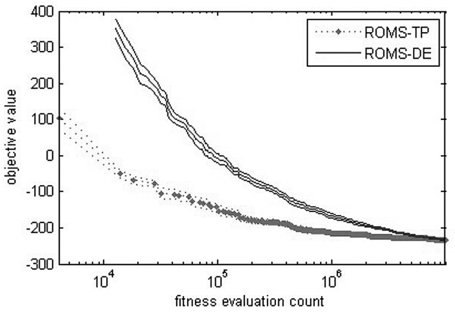

Fig6(d). F_1-20-8-2-f_{avg}

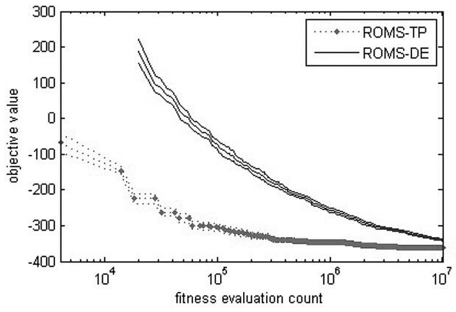

Fig6(e). F_1-20-10-2-f_{avg}

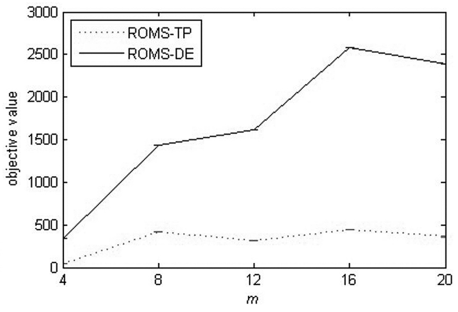

Fig7(a). F_1-m-2-2-f_{avg}

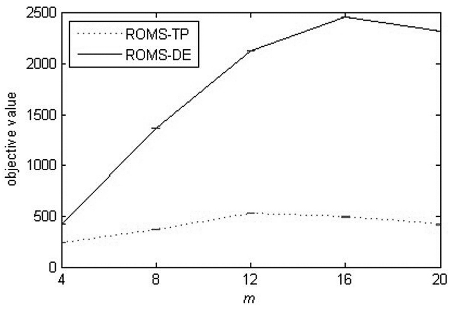

Fig7(b). F_2-m-2-2-f_{avg}

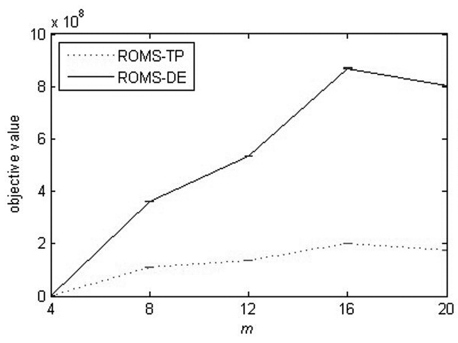

Fig7(c). F_3-m-2-2-f_{avg}

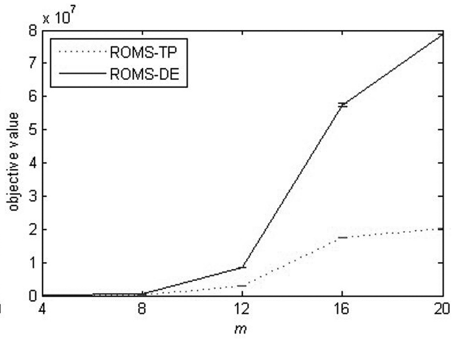

Fig7(d). F_6-m-2-2-f_{avg}

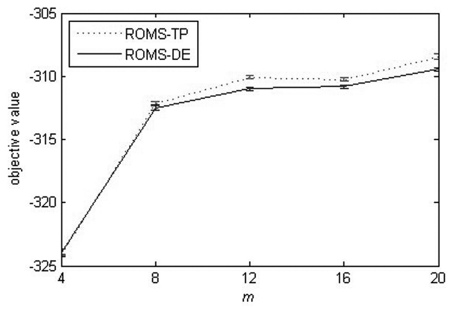

Fig7(e). F_{10}-m-2-2-f_{avg}

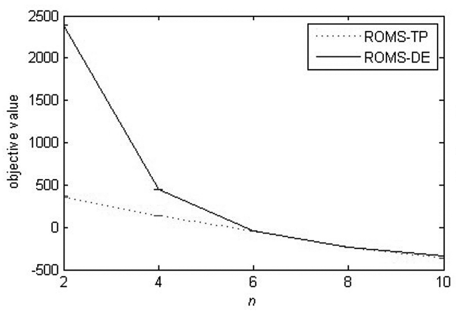
Fig8(a). F_1-20-n-2-f_{avg}

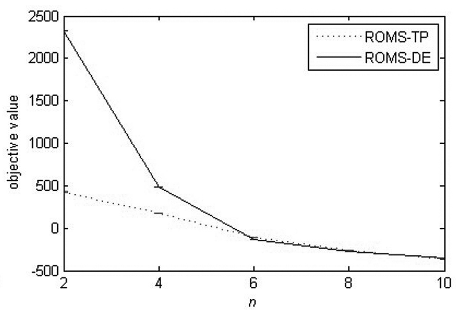
Fig8(b). F_2-20-n-2-f_{avg}

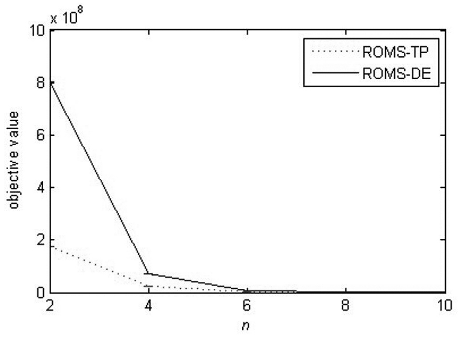
Fig8(c). F_3-20-n-2-f_{avg}

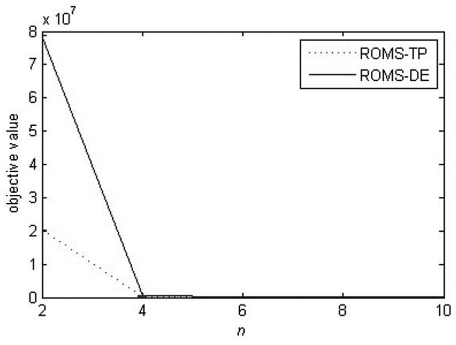
Fig8(d). F_6-20-n-2-f_{avg}

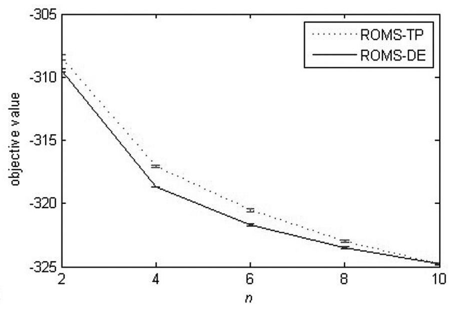
Fig8(e). F_{10}-20-n-2-f_{avg}

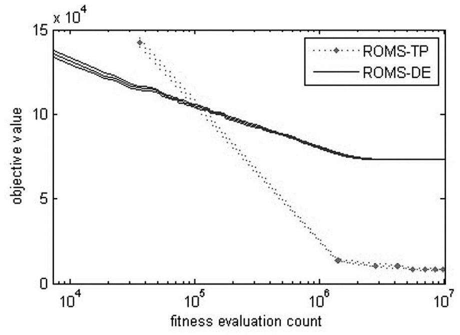
Fig9. F_1-12-2-30-f_{avg}

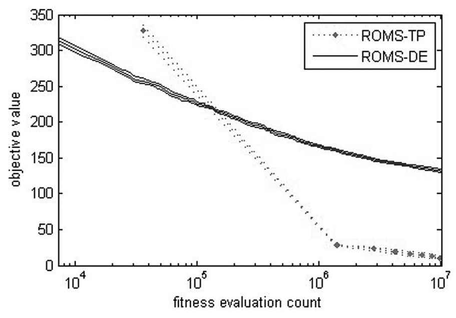
Fig10. F_{10}-12-2-30-f_{avg}

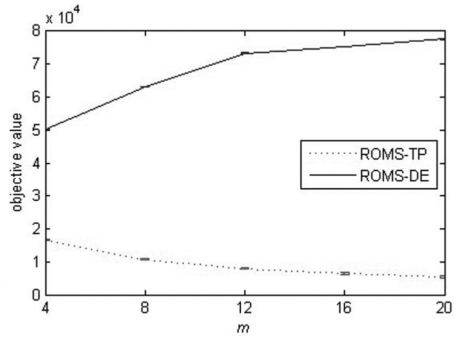
Fig11(a). F_1-m-2-30-f_{avg}

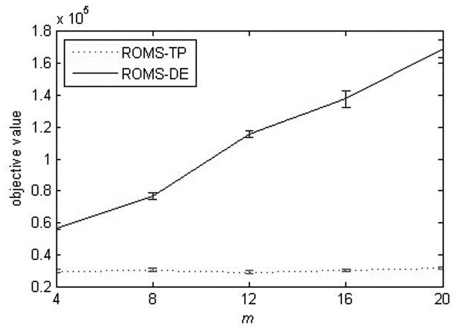
Fig11(b). F_2-m-2-30-f_{avg}

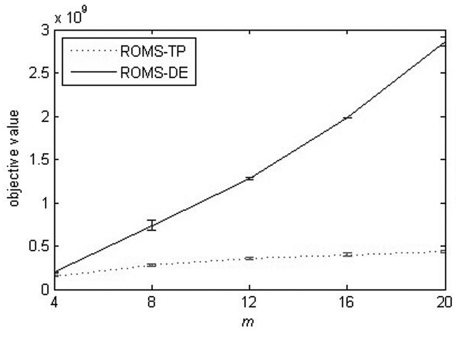
Fig11(c). F_3-m-2-30-f_{avg}

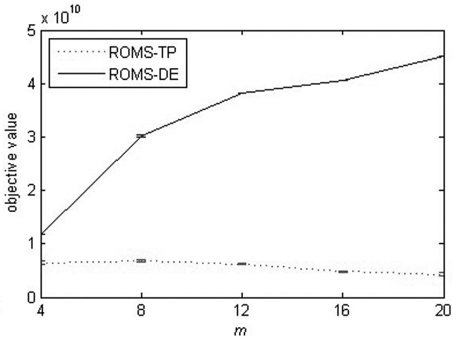
Fig11(d). F_6-m-2-30-f_{avg}

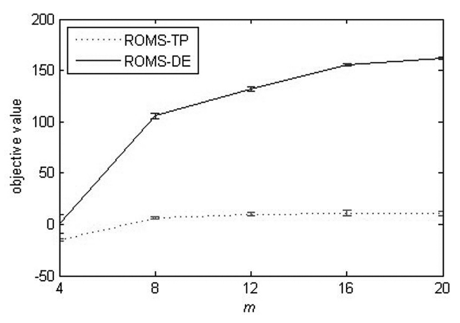
Fig11(e). F_{10}-m-2-30-f_{avg}

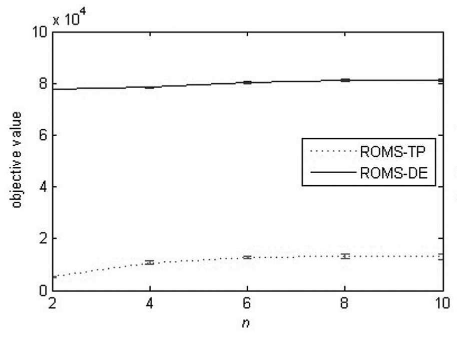

Fig12(a). F_1-20-n-30-f_{avg}

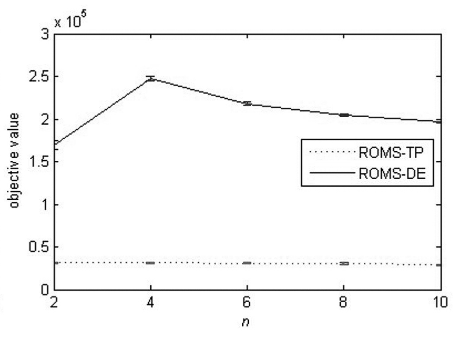

Fig12(b). F_2-20-n-30-f_{avg}

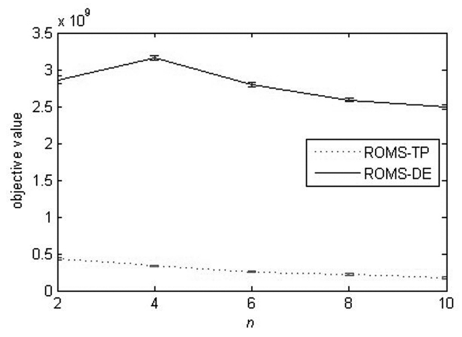

Fig12(c). F_3-20-n-30-f_{avg}

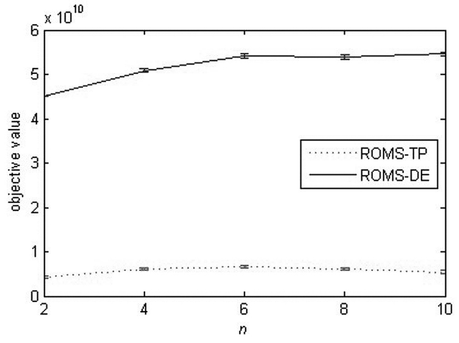

Fig12(d). F_6-20-n-30-f_{avg}

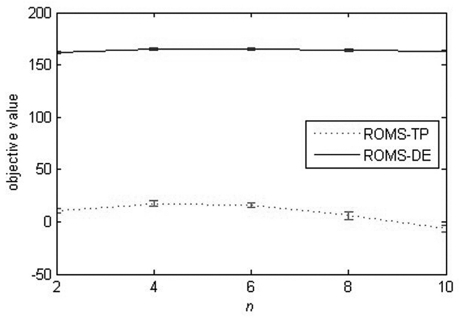

Fig12(e). F_{10}-20-n-30-f_{avg}

results on F_{10} are somehow abnormal when compared to the results on other test functions. Since high multimodality is the most distinct feature of F_{10}, it is suspected that the degree of multimodality has an impact on the relative performance of the two algorithms. Specifically, ROMS-DE seems to be more suitable for handling highly multimodal functions.

Optimization dynamics for f_{avg}, $D = 30$: At a higher dimension $D = 30$, when n is fixed, the big picture looks similar with different m values and on different test functions. Figs. 9 and 10 give the results corresponding respectively to Figs. 4 and 5. Pay attention that, for the ROMS-TP curve in the two figures, the fitness evaluation cost and objective value decrease of the first step are so low compared to those of the second step that the sample point for the second step is visually undistinguishable from the sample point for the first step in the next iteration; they just overlapped. Compared to results in 2 dimensions, the difference is mainly seen on F_{10}. Here, F_{10} does not distinguish itself from other test functions, and the results are just like those on other functions without any particularity. The success of the second step may due to the increase in the population size, which according to the parameter settings (see Table 3) is $5\times D$. When m is fixed, the dynamics of ROMS-TP basically remains the same for different n values while the optimization process of ROMS-DE is becoming more and more less well developed with increasing n due to an increasing population size (just like the case in 2 dimensions).

Scalability for f_{avg}, $D = 30$: Fig. 11 gives the results of scalability test with respect to m in 30 dimensions, which clearly show that ROMS-TP has better scalability. Fig. 12, on the other hand, gives the results in terms of n, and no clear scalability difference is observed. In 30 dimensions, ROMS-TP has always got a better final result on all experiments including the ones on F_{10}, which is an exception in 2 dimensions.

Optimization dynamics for f_{wst}, $D = 2$: In the remainder of this subsection, the results obtained with measure f_{wst} are presented. The dynamics picture looks basically the same for different m values and on different test functions. Figs. 13 and 14 show respectively the results corresponding to Figs. 4 and 5. For ROMS-TP, contribution of the first step to objective value decrease is limited compared to that of the second step on all test functions. Fig. 15 shows the results with an increasing n. The optimization process of ROMS-DE is becoming less well developed while the curve of ROMS-TP is becoming smoother. However, the performance difference of the two algorithms seems to be enlarging. The two curves are more deviated for larger n values.

Scalability for f_{wst}, $D = 2$: Fig. 16 shows the results of scalability test with respect to m. The curves are so close that we are unable to draw a clear conclusion. Fig. 17 shows the results with respect to n. Despite the curves are close, ROMS-TP has better scalability.

Optimization dynamics for f_{wst}, $D = 30$: As in 2 dimensions, the results look similar for different m values on all test functions. Figs. 18 and 19 give the results corresponding to Figs. 13 and 14 respectively. Again, for the ROMS-

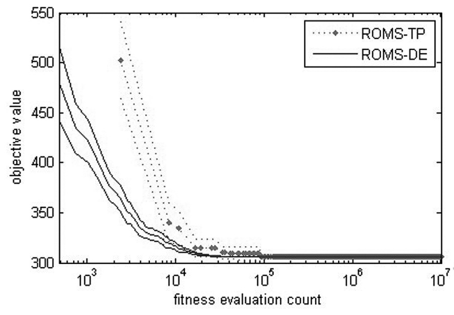

Fig13. F_1-12-2-2-f_{wst}

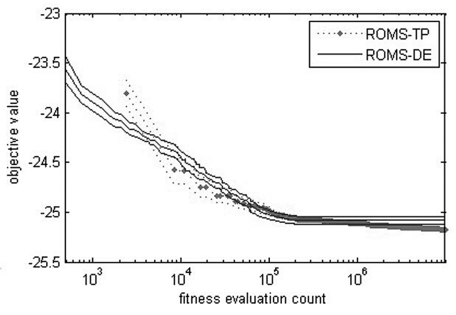

Fig14. F_{10}-12-2-2-f_{wst}

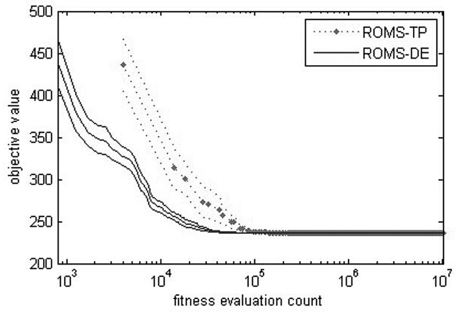

Fig15(a). F_1-20-2-2-f_{wst}

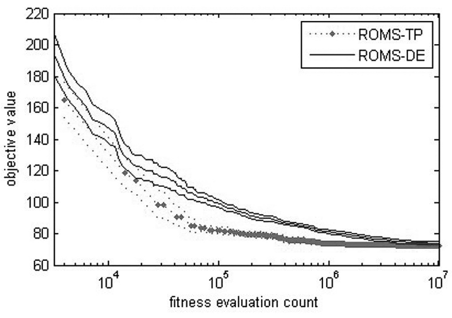

Fig15(b). F_1-20-4-2-f_{wst}

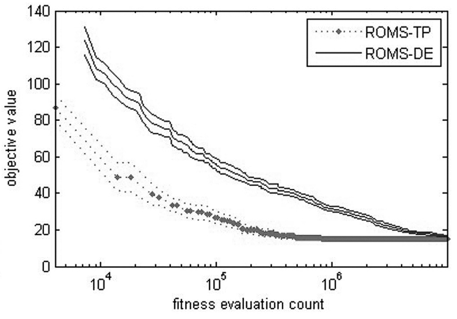

Fig15(c). F_1-20-6-2-f_{wst}

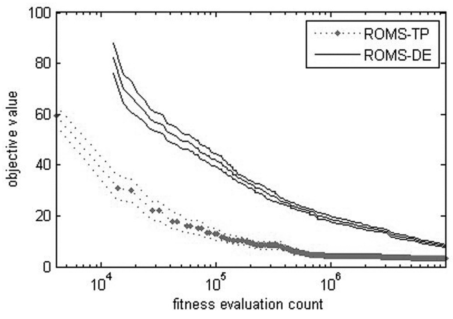

Fig15(d). F_1-20-8-2-f_{wst}

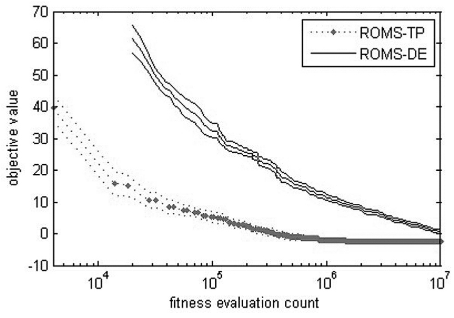

Fig15(e). F_1-20-10-2-f_{wst}

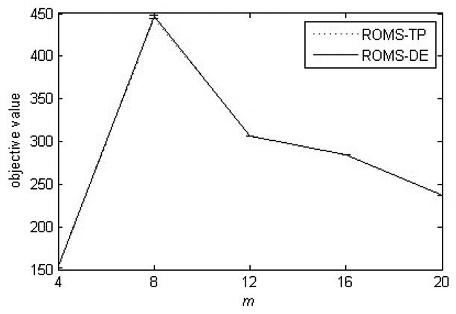

Fig16(a). F_1-m-2-2-f_{wst}

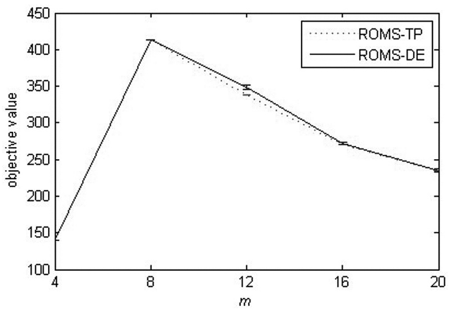

Fig16(b). F_2-m-2-2-f_{wst}

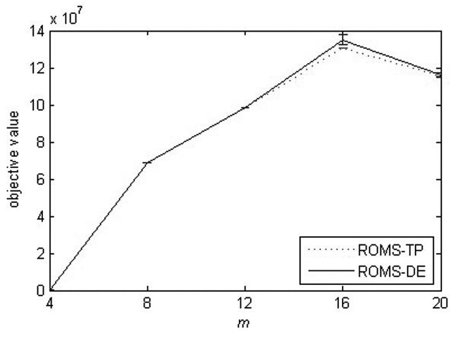

Fig16(c). F_3-m-2-2-f_{wst}

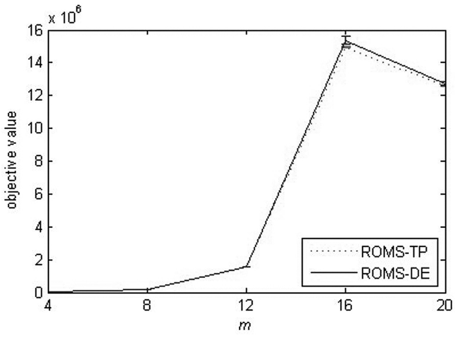

Fig16(d). F_6-m-2-2-f_{wst}

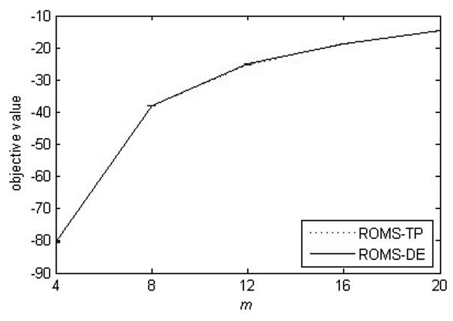

Fig16(e). F_{10}-m-2-2-f_{wst}

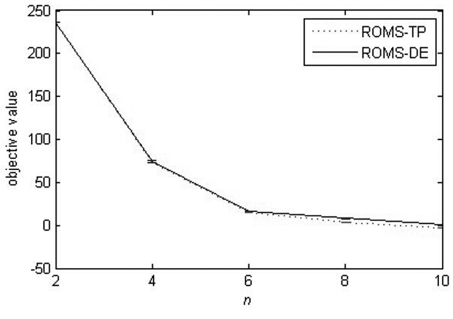

Fig17(a). F_1-20-n-2-f_{avg}

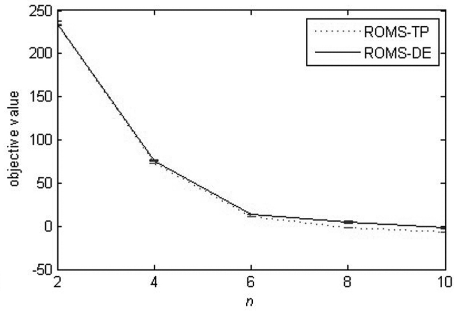

Fig17(b). F_2-20-n-2-f_{avg}

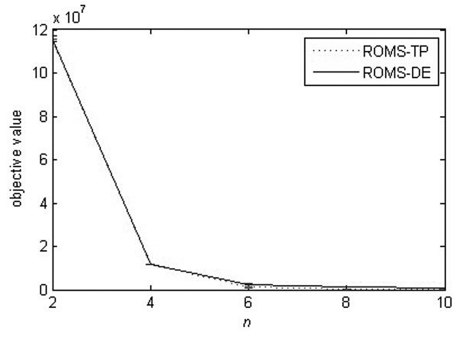

Fig17(c). F_3-20-n-2-f_{avg}

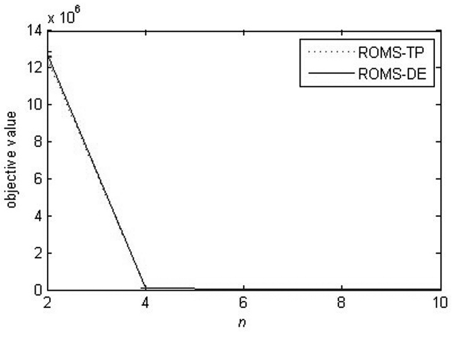

Fig17(d). F_6-20-n-2-f_{avg}

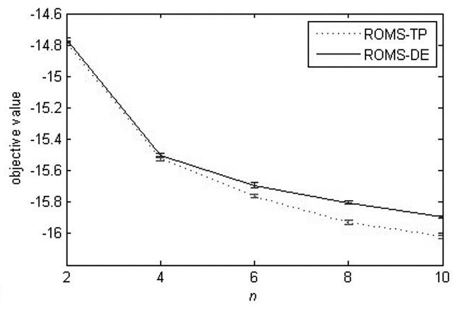

Fig17(e). F_{10}-20-n-2-f_{avg}

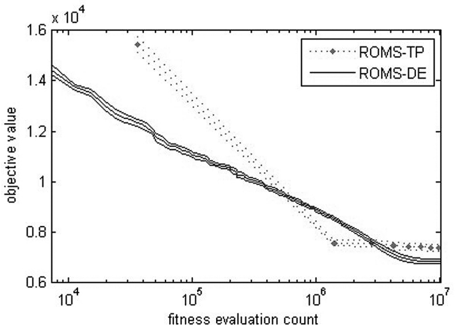

Fig18. F_1-12-2-30-f_{avg}

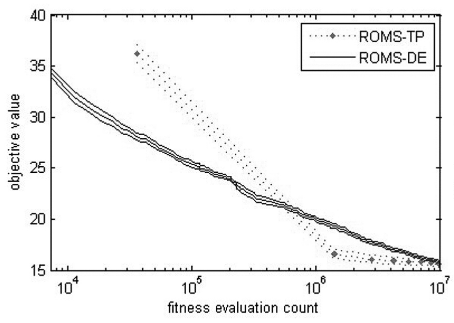

Fig19. F_{10}-12-2-30-f_{avg}

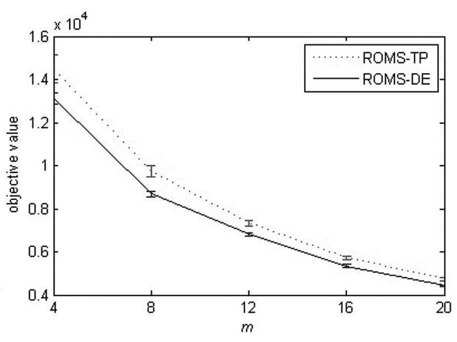

Fig20(a). F_1-m-2-30-f_{avg}

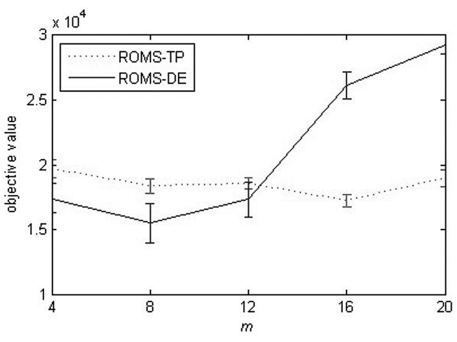

Fig20(b). F_2-m-2-30-f_{avg}

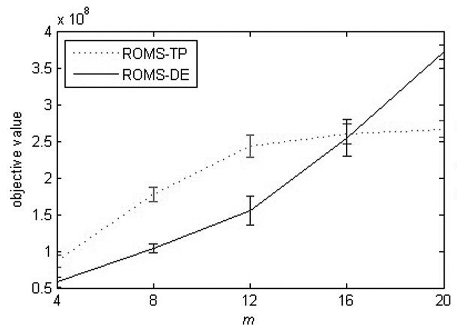

Fig20(c). F_3-m-2-30-f_{avg}

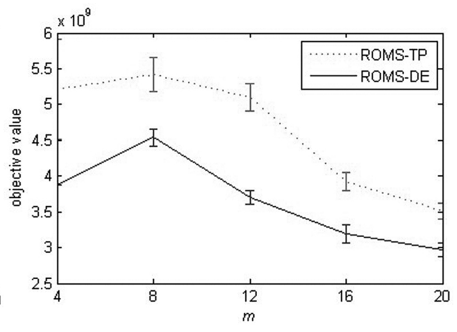

Fig20(d). F_6-m-2-30-f_{avg}

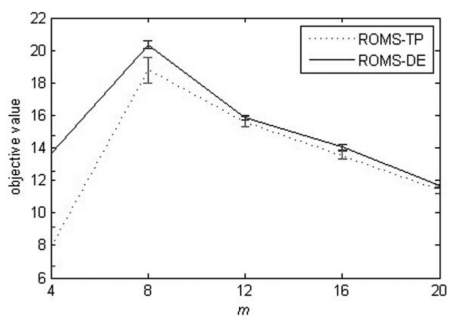

Fig20(e). F_{10}-m-2-30-f_{avg}

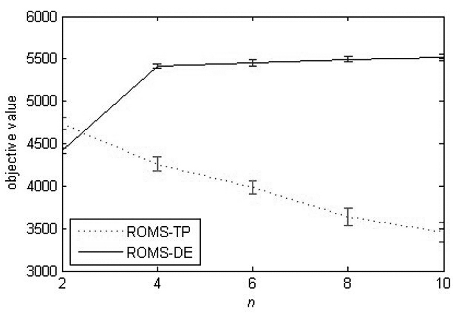

Fig21(a). F_1-20-n-30-f_{avg}

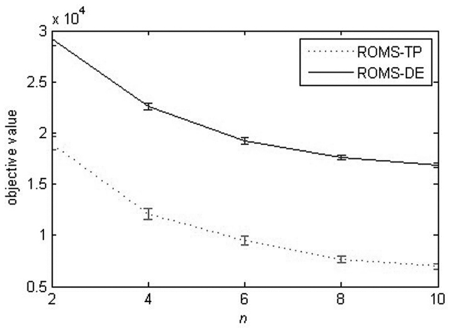

Fig21(b). F_2-20-n-30-f_{avg}

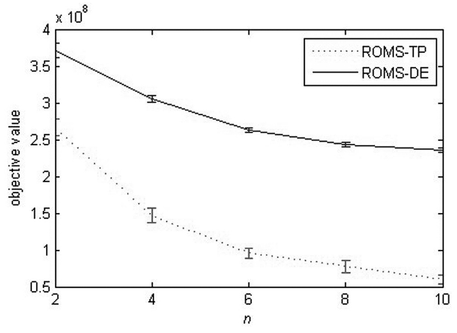

Fig21(c). F_3-20-n-30-f_{avg}

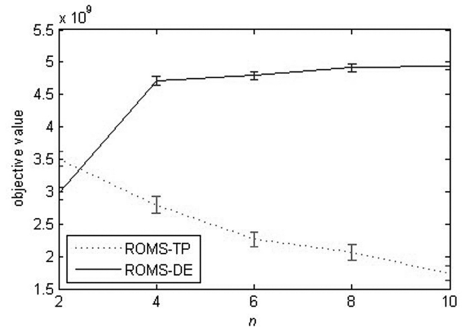

Fig21(d). F_6-20-n-30-f_{avg}

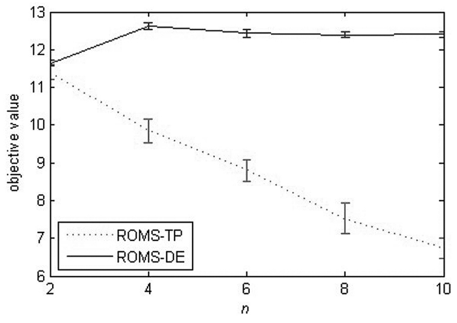

Fig21(e). F_{10}-20-n-30-f_{avg}

TP curve, the sample point for the second step is visually undistinguishable from the sample point for the first step in the next iteration.

Scalability for f_{wst}, $D = 30$: Fig.20 gives the results of scalability test with respect to m, which show that ROMS-TP has better scalability on all test functions except F_{10} where the opposite seems to hold. Fig. 21 gives the results with respect to n, which clearly suggest that ROMS-TP has better scalability on all test functions including F_{10}. With f_{wst} in 30 dimensions, ROMS-DE has got a better final result than that of ROMS-TP in some cases. The advantage of ROMS-TP is somehow weakened.

VII. Conclusions

In this work, a new formulation for robust optimization is proposed. The goal is to optimize for a variety of environmental conditions which are modelled by a set of functions. Following a traditional robust optimization formulation, a single solution is sought to cope with all different conditions, which characterizes an extremely high robustness requirement. Since robustness and performance are generally two conflicting goals, this traditional formulation may fail with way too compromised performance in some applications. To address this issue, we propose a generalized formulation, namely robust optimization with multiple solutions, in which multiple solutions, are sought to cope with the varied conditions. Each solution serves some subset of the conditions and the whole condition set is covered by the solution set. By tuning the desired number of solutions, the balance between robustness and performance can be flexibly adjusted.

The formulation presents us a new kind of optimization problem with new difficulty. To solve it, two fundamental observations revealing insights into the problem are first made. $Observation 1$ is about computing an optimal mapping from a given solution set, while $Observation 2$ is about computing an optimal solution set from a given condition set partition. Based on the observations, two general approaches are then proposed. The first approach is by direct evolutionary optimization. Taking advantage of the first observation, the original mixed-integer optimization problem is reduced to a real-value one which is much easier to deal with. With a unified view on the two observations, an optimization loop is discovered. The second approach is built on the basic idea that optimization can be achieved by repeating this loop. Our proposed approaches apply to the robustness measure of both average and the worst performance.

With specific instantiations, denoted by ROMS-DE and ROMS-TP respectively, of the two approaches, an empirical study is performed. The experiments are done using various objective functions from CEC-2005, with different problem parameters (number of conditions and desired number of solutions), and in different dimensions (2 and 30). To introduce the notion of environmental parameters, the original objective functions are parameterized with a shifting vector and an orthogonal matrix, which are randomly generated for each condition. According to the results, both algorithms have their wins and losses. To sum up, ROMS-TP has a better overall performance. As to future work, theoretical analysis of the proposed approaches is encouraged. We are also eager to see some real-world applications.

Acknowledgment

This research has been partially supported by IEEE Computational Intelligence Society Graduate Student Research Grant 2015.

References

[1] A. Ben-Tal and A. Nemirovski, "Robust solutions of linear programming problems contaminated with uncertain data," *Mathematical programming*, vol. 88, no. 3, pp. 411–424, 2000.

[2] H.-G. Beyer and B. Sendhoff, "Robust optimization–a comprehensive survey," *Computer methods in applied mechanics and engineering*, vol. 196, no. 33, pp. 3190–3218, 2007.

[3] J. W. Kruisselbrink *et al.*, *Evolution strategies for robust optimization*. Leiden Institute of Advanced Computer Science (LIACS), Faculty of Science, Leiden university, 2012.

[4] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-a survey," *Evolutionary Computation, IEEE Transactions on*, vol. 9, no. 3, pp. 303–317, 2005.

[5] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski, *Robust optimization*. Princeton University Press, 2009.

[6] R. R. Chan and S. D. Sudhoff, "An evolutionary computing approach to robust design in the presence of uncertainties," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 6, pp. 900–912, 2010.

[7] I. Paenke, J. Branke, and Y. Jin, "Efficient search for robust solutions by means of evolutionary algorithms and fitness approximation," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 4, pp. 405–420, 2006.

[8] K. Deb, S. Gupta, D. Daum, J. Branke, A. K. Mall, and D. Padmanabhan, "Reliability-based optimization using evolutionary algorithms," *Evolutionary Computation, IEEE Transactions on*, vol. 13, no. 5, pp. 1054–1074, 2009.

[9] A. M. Cramer, S. D. Sudhoff, and E. L. Zivi, "Evolutionary algorithms for minimax problems in robust design," *Evolutionary Computation, IEEE Transactions on*, vol. 13, no. 2, pp. 444–453, 2009.

[10] D. H. Loughlin and S. R. Ranjithan, "Chance-constrained genetic algorithms," in *Proc. Genetic Evol. Comput. Conf*, 1999, pp. 369–376.

[11] J. Branke, "Reducing the sampling variance when searching for robust solutions," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, L. Spector, E. D. Goodman, A. Wu, W. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, Eds. San Francisco, California, USA: Morgan Kaufmann, 7-11 July 2001, pp. 235–242.

[12] K. Deb, "Recent developments in evolutionary multi-objective optimization," in *Trends in multiple criteria decision analysis*. Springer, 2010, pp. 339–368.

[13] Y. Jin and B. Sendhoff, "Trade-off between performance and robustness: an evolutionary multiobjective approach," in *Evolutionary Multi-Criterion Optimization*. Springer, 2003, pp. 237–251.

[14] A. Shapiro, D. Dentcheva *et al.*, *Lectures on stochastic programming: modeling and theory*. SIAM, 2014, vol. 16.

[15] H. Agarwal, "Reliability based design optimization: formulations and methodologies," Ph.D. dissertation, 2004.

[16] D. G. Robinson, "A survey of probabilistic methods used in reliability, risk and uncertainty analysis: analytical techniques i," *Sandia National Lab, Report SAND98*, vol. 1189, 1998.

[17] D. Bertsimas and M. Sim, "The price of robustness," *Operations research*, vol. 52, no. 1, pp. 35–53, 2004.

[18] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 32–49, 2011.

[19] S. Das, S. Maity, B.-Y. Qu, and P. N. Suganthan, "Real-parameter evolutionary multimodal optimizationa survey of the state-of-the-art," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 71–88, 2011.

[20] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[21] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95–99, 1988.

[22] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization," *KanGAL report*, vol. 2005005, 2005.

[23] G. Strang, *Introduction to Linear Algebra (4th ed.)*. Wellesley-Cambridge Press Wellesley, MA, 2009.

[24] P. Diaconis and M. Shahshahani, "The subgroup algorithm for generating uniform random variables," *Probability in the Engineering and Informational Sciences*, vol. 1, no. 01, pp. 15–32, 1987.