SINGLE AND MULTI-OBJECTIVE OPTIMIZATION USING EVOLUTIONARY COMPUTATION

KALYANMOY DEB
Department of Mechanical Engineering
Indian Institute of Technology Kanpur
Kanpur, PIN 208016, India
Email: deb@iitk.ac.in
http://www.iitk.ac.in/kangal/deb.htm

KanGAL Report Number 2004002

The use of evolutionary computation (EC) in the solution of optimization problems of hydroinformatics is not new. Many such problems having single or multiple objectives are now routinely solved using different EC methodologies. Through this invited paper for the proceedings of the Hydroinformatics 2004 conference, I am delighted to share some recent advances in the area of evolutionary computing with the researchers and practitioners to this field. Some critical issues, such as design of an efficient evolutionary algorithm (EA), an efficient constraint handling procedure, scalability issue of EAs, and multi-objective optimization are dealt in this paper. The discussion of the topics and subsequent engineering and numerical case studies presented in this paper should be useful to EA and non-EA hydroinformaticians alike.

INTRODUCTION

The field, known collectively as Hydroinformatics, involves design of water distribution networks and systems [34, 1, 6, 26, 30], water supply management [5], water resources [33, 45], watershed water quality management [20, 42], water pollution control [41], ground water monitoring [21, 23, 7, 39], waste water management [4], irrigation scheduling [37], etc. Each of these topics involves one or more goals or objectives and are posed as single or multi-objective optimization problems in the literature and solved. The journal of Hydroinformatics and other domain-specific journals and conference proceedings describe innumerable such optimization case studies. In many of such studies, an *evolutionary algorithm* of some sort is often used. Although the author of this paper is not an expert in the area of hydroinformatics, a look at some of the above problems will convince an optimization specialist that these optimization problems possess a number of difficulties and complexities which are beyond the scope of classical point-by-point search and optimization algorithms. Non-existence of gradients of objectives and constraints, discreteness in the decision variables, uncertainities in evaluating objectives and constraints, existence of multiple conflicting objectives etc. are commonplace in such problems. It is

not at all surprising that the hydroinformaticians are finding unorthodox and non-classical optimization algorithms, such as evolutionary algorithms, useful in solving such problems.

In this paper, we describe evolutionary algorithms as a population-based optimization tool and show their efficacies in handling different vagaries of optimization problems, such as in handling mixed type of variables, non-linear constraints, large-sized variables, presence of multiple objectives etc. Although we do not discuss any hydroinformatics problem per se (apologizes for not being an expert in the domain), appropriate engineering design and optimization problems are chosen to illustrate the use of EAs on similar other problems. On this account, the reader of this paper may find it to be useful in knowing the strengths and weaknesses of EAs, which will allow a proper application of EAs to an optimization problem.

EVOLUTIONARY ALGORITHMS (EA)

In principle, evolutionary algorithms (EAs) mimic natural evolutionary concepts to constitute search and optimization procedures. In reality, EAs are different from classical search and optimization procedures in a variety of ways. A clear distinction between the two is that an EA uses a population of solutions in each iteration and a classical method uses only one solution. Because of this reason, we shall call the EA method a *population-based* approach and the classical method a *point-by-point* approach. In an earlier study, Goldberg [28] highlighted some aspects of designing a competent EA for hydroinformaticians. In this section, we shall present a generic algorithm-generator, which can be used to develop an efficient EA with desired properties [13] and suggest what features make them unique and useful in solving engineering optimization problems, including problems from hydroinformatics.

A Population-Based Optimization Algorithm Generator

As the name suggests, a population-based optimization algorithm begins its search with a population of guess solutions. Thereafter, in each iteration the population is updated by using a population-update algorithm. We assume that the algorithm at iteration t has a set of points $B^{(t)}$ (with $N = |B^{(t)}|$). At the end of each iteration, this set is updated to a new set $B^{(t+1)}$ by using four user-defined plans. For brevity, here we drop the superscript denoting the iteration counter from the sets.

Population-Based-Optimization-Algorithm(SP, GP, RP, UP)

Step 1 Choose μ solutions (the set P) from B using a *selection plan* (SP).

Step 2 Create λ solutions (the set C) from P using a *generation plan* (GP).

Step 3 Choose r solutions (the set R) from B for replacement using a *replacement plan* (RP).

Step 4 Update these r members by r solutions chosen from a comparison set of R, P, and C using a *update plan* (UP).

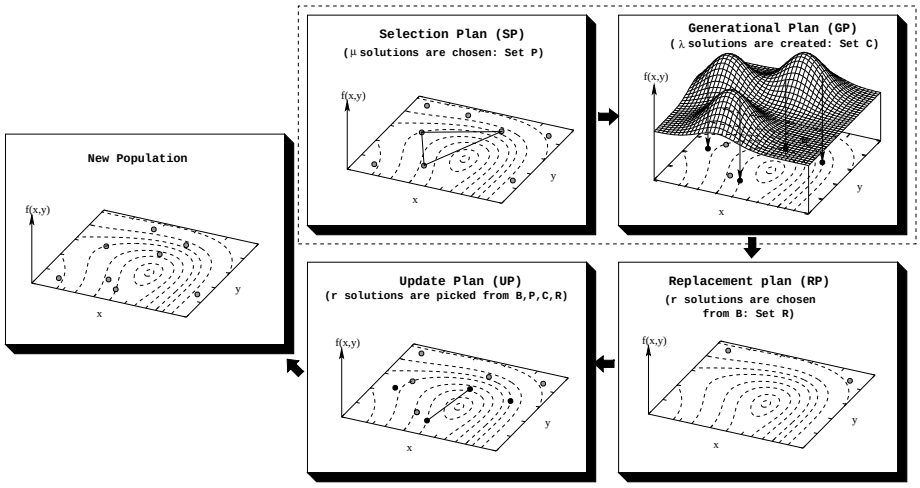

Figure 1: A sketch of one iteration of the proposed population-based optimization procedure. For some realizations, the SP and GP steps can be applied multiple times before proceeding to the RP and UP plans.

Since by choosing a plan for each step one develops a new optimization algorithm, we call the above procedure an algorithm-generator. Figure 1 shows a sketch of the proposed generic procedure. The first step chooses μ solutions from the solution bank B (all solutions in the top-left plot) for their use in creating new solutions in the second step. Thus, the selection plan (SP) for choosing these μ solutions must emphasize the *better* solutions of B. The chosen solutions ($\mu = 3$ in the figure) are shown as dark-outlined points on the top-left plot and are also joined by lines. Clearly, there are two ways to choose a good SP. A set of μ solutions can be chosen either by directly emphasizing the better solutions in B or by de-emphasizing the worse solutions of B. The algorithm can be made somewhat 'greedy' by always including a copy of the 'best' solution of B in P.

In the second step, λ new solutions are created from the chosen set P by using a generation plan GP. One convenient procedure would be to choose a probability distribution function (PDF) using the set P and create λ solutions. Figure 1 shows a typical probability distribution in which solutions around the members of P are emphasized. Members of the set C ($\lambda = 4$ in the figure) are shown in solid circles in the figure. This is the most crucial step of the proposed algorithm-generator. Although for generating each new solution a different PDF can be used thereby requiring the need to use SP-GP steps (shown in a dashed box) many times before proceedings further, for simplicity one PDF can be chosen to create all λ new solutions.

In the third step, r solutions are chosen from the solution bank B for a replacement using the newly created solutions. Here, different replacement plans are possible. The RP can simply choose r solutions at random or include some or all

members of P to ensure diversity preservation. In the figure, r (= 2 in the figure) worst solutions from B are chosen as members of R.

In the fourth step, the r chosen members are updated by r members chosen from R, P, and C by using an update plan (UP). It is intuitive that the UP must emphasize the better solutions of $R \cup P \cup C$ in this step. However, the r slots can also be updated directly from members of C alone or from a combined population $R \cup C$ or $P \cup C$. In the former case, the population update algorithm does not have the elite-preserving property. To really ensure elite-preservation, the RP should choose the best solutions of B and a combined P and C set needs to be considered in Step 4. In the figure, the best r solutions (joined by a line) from the $R \cup P \cup C$ are chosen and the originally chosen r solutions in Step 3 are replaced by these r solutions.

Although the above procedure is not the only possible way to describe a population-based algorithm-generator, it certainly makes a *functional decomposition* of the salient tasks needed in a good search and optimization algorithm. The selection plan emphasizes better solutions from a population, the generational plan uses these solutions to explore the search region around them and create new solutions, the replacement plan picks potentially bad solutions from the original population to be replaced, and finally the update plan completes the replacement task with the newly created solutions. As described above, each plan controls a crucial property of the resulting algorithm. By properly choosing a strategy for each plan, the resulting optimization algorithm can be made *greedy* to mainly solve unimodal problems, can have a global perspective for targeting to find the global optimum, or can be made to be a monotonically non-deteriorating algorithm to have sustained improvement in the solution quality.

The procedure of using the above algorithm-generator for developing a new optimization algorithm is as follows:

1. Choose a suitable plan for each step (SP, GP, RP, and UP).
2. Perform a parametric study to discover optimal algorithm parameters, such as N, μ, λ and r.

The original study [13] used the above algorithm-generator to describe a number of population-based EAs and classical methods. The above dichotomy of different functionalities of a population-based optimization algorithm will be useful in creating a new customized EA having desired properties and having essential features needed in a competent optimization algorithm.

EAs FOR ENGINEERING OPTIMIZATION

Engineering optimization problems involve a number of characteristics, which make them difficult to be solved with a required level of satisfaction by using classical point-by-point numerical optimization algorithms:

- Existence of mixed type of variables (such as Boolean, discrete, integer and real).

- Existence of non-linear constraints.
- Existence of multiple conflicting objectives.
- Existence of multiple optimum (local and global).
- Existence of stochasticities and uncertainties in describing the optimization problem.

We describe the first three aspects and their possible handling using an evolutionary algorithm in the rest of the paper. Although we do not undermine the importance of handling the rest of the aspects mentioned above, we simply state that because of the stochastic nature of the working of an EA, they are more suitable for handling the multi-modality and uncertainty issues associated with engineering optimization problems than the classical point-by-point approaches. Goldberg [27] addressed these issues in his seminal book.

Handling Mixed Variables

Many engineering optimization problems involve mixed variables meaning that the decision variable vector has some components which can only take integer or discrete values and some which can take any real value. The classical optimization procedures use some *fix-ups* to handle integer or discrete restrictions, simply because the the concept of gradients or local neighborhoods may not exist in such problems. Evolutionary algorithms have been comparatively more successful in solving these problems. In the following, we describe how EAs are more suitable in solving such problems.

In an EA, a mixed variable vector can be represented by using a mixed coding – discrete variables can be coded using a binary substring and continuous variables can be represented directly. Thereafter, mixed GA operators can be used to create new offspring solutions. The variables represented using a binary substring, the usual bit-operated crossover and mutation operators [27, 32] can be used and for real-parameter variables, a real-coded recombination and mutation operators [11, 31] can be used. This is possible because between any two parent solutions, a variable-wise crossover/recombination means recombining variables of the same type. To illustrate the concept of a mixed coding, we consider the design of a cantilever beam having four variables:

$$((1) \quad 14 \quad 23.457 \quad (101))$$

The first variable (a Boolean variable) can take only one of two values (either `1` or `0`). The value `1` represents a circular cross-section and the value `0` represents a square cross-section of the cantilever beam. The second variable represents the diameter of the circular section if the first variable is a `1` or the side of the square if the first variable is a `0`. This variable is discrete, since arbitrary section sizes are not usually available for fabrication. The third variable represents the length of the cantilever beam. Thus, it could be a real-parameter variable. The fourth variable is a discrete variable representing the material, which can take one of 8

pre-specified materials. For simplicity, a set of three binary digits (totalling 2^3 or 8 values) may be used to code this variable. Thus, the above string represents a cantilever beam made of the 5-th material from a prescribed list of 8 materials having a circular cross-section with a diameter 14 mm and having a length of 23.457 mm. With the above flexibility in design representation, any combination of cross sectional shape and size, material specifications, and length of the cantilever beam can be represented. This flexibility in the representation of a design solution is not possible with classical optimization methods. Since each design variable is allowed to take only permissible values (for example, no hexagonal shape will be tried or no unavailable diameter of a circular cross-section will be chosen), the computational time for searching of the optimal problem is also expected to be substantially less.

With such a representation scheme, a standard reproduction operator can still be used. However, for crossover and mutation operations, the respective genetic operator described above can be suitably used. For real variables, the simulated binary crossover (SBX) [14] and the polynomial mutation operator [17] and for integer variables represented by binary substrings, standard binary crossover and mutation operators [27, 32] can be used. For unbounded variables, the SBX operator applied between two parent values p_1 and p_2 is as follows:

Step 1: Choose a random number $u \in [0, 1)$.

Step 2: Calculate β_q using

$$\beta_q = \begin{cases} (2u)^{\frac{1}{\eta_c+1}}, & \text{if } u \leq 0.5; \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\eta_c+1}}, & \text{otherwise.} \end{cases}$$

Step 3: Compute two offspring solutions as follows:

$$\begin{aligned} c_1 &= 0.5\left[(1+\beta_q)p_1 + (1-\beta_q)p_2\right], \\ c_2 &= 0.5\left[(1-\beta_q)p_1 + (1+\beta_q)p_2\right]. \end{aligned}$$

Here, η_c is the distribution index, which controls the spread of offspring solutions and is usually kept in the range $[1, 10]$. Similarly, the polynomial mutation operator perturbs the solution c (with a random number $r \in [0, 1]$) to create a new solution c', as follows: $c' = c + (x^{(U)} - x^{(L)})\bar{\delta}$, where $x^{(U)}$ and $x^{(L)}$ are upper and lower bounds of the variable x and

$$\bar{\delta} = \begin{cases} (2r)^{1/(\eta_m+1)} - 1, & \text{if } r < 0.5, \\ 1 - [2(1-r)]^{1/(\eta_m+1)}, & \text{if } r \geq 0.5. \end{cases}$$

Here, η_m is the distribution index controlling the spread of mutated solution c' from c and is kept in the range $[10, 100]$.

It is important to highlight here that besides the real-parameter genetic algorithms, there exists other real-parameter EAs – such as evolution strategies [43, 2], differential evolution [44], and evolutionary programming [25, 24] – for the purpose.

A recent study [3] compared a number of real-parameter GAs and evolution strategies and concluded that for an appropriate setting of a crucial parameter in each algorithm, their performances can be more or less equivalent.

Another study [16] suggested a steady-state population-based approach (G3) with a vector-wise recombination operator (parent-centric crossover of PCX) for real-parameter optimization and systematically compared its performance with evolution strategies, differential evolution and the classical quasi-Newton method (BFGS) method [38] on a number of test problems. It is observed that a specific self-adaptive ES (CMA-ES) and the suggested G3-PCX perform better than other evolutionary methods and also better than the BFGS method on difficult test problems. Overall, this study concluded that the evolutionary algorithms are competitive and are ideal optimization tools for solving complex optimization tasks. To show the scalability of G3PCX to large-sized problem-solving, we show the performance of the G3-PCX on the Schwefel's function in Figure 2:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2 . \tag{1}$$

Up to a 500-variable problem, the proposed method shows a sub-quadratic complexity in terms of required evaluations to reach close (with 10^{-20} accuracy) to the true optimum.

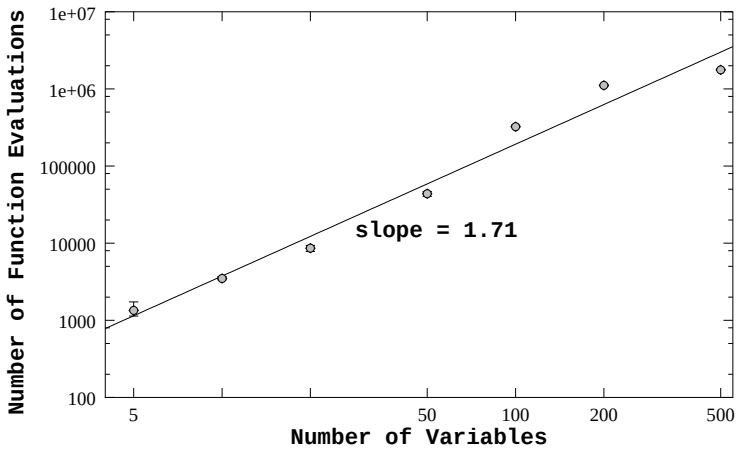

Figure 2: Scale-up study for the Schwefel's function using the modified G3 model and the PCX recombination operator. Ten runs on each problem show not much variation in the overall requirement of function evaluations.

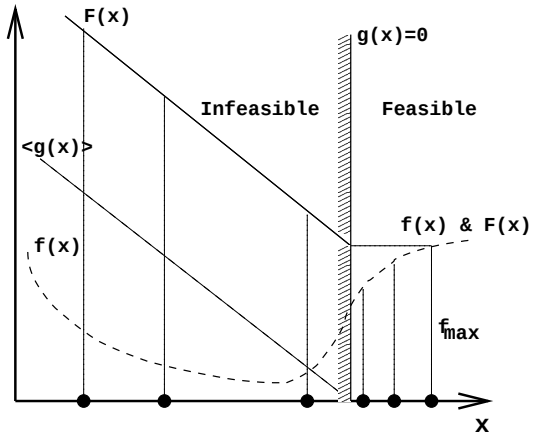

Figure 3: A population-based constraint handling strategy which does not require any penalty parameter.

Handling Constraints

Handling non-linear constraints, often arise in engineering optimization problems, is a major difficulty to the classical point-by-point approaches. A common practice is to use a penalty function approach [9, 38, 40], in which a *normalized* constraint violation is multiplied by a penalty parameter R and added to the objective function. The main difficulty arises in choosing the penalty parameter R. With such a method, several questions arise: (i) Should R be kept constant or varied with iterations? (ii) How to choose an appropriate R? (iii) How sensitive is R to the obtained solution? Here, we discuss a population-based approach which completely avoids using any penalty parameter.

Using the objective function and the constraints, we define the following composite *fitness* function for any solution $\mathbf{x}$[10, 11]:

$$F(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{if } \mathbf{x} \text{ is feasible;} \\ f_{\max} + \text{CV}(\mathbf{x}), & \text{otherwise.} \end{cases} \tag{2}$$

Here, $f_{\max}$ is the objective function value of the worst feasible solution in the population and $\text{CV}(\mathbf{x})$ is the overall normalized constraint violation of solution $\mathbf{x}$. Figure 3 shows the construction procedure of fitness function $F(x)$ from the objective function $f(x)$ and constraint function $g(x)$ for a single-variable objective function. Since all feasible solutions have zero constraint violation and all infeasible solutions are evaluated according to their constraint violations only, both the objective function value and constraint violation are not combined in any solution in the population. Thus, there is no need to have any penalty parameter R for this approach. It is important to realize that such a penalty-parameter-less strategy is only applicable to a population-based approach. This is because it requires to divide the population into two: feasible and infeasible sets. The fitness function depends on the feasi-

ble and infeasible population members and hence it is a dynamic one. Since in a point-by-point optimization approach there is only member in each iteration, such a penalty-parameter-less scheme is not possible. Moreover, the approach is also pragmatic. Since infeasible solutions are not to be recommended for use, there is no real reason for one to find the objective function value for an infeasible solution.

Knowledge-Based EAs

The above methodologies for an evolutionary algorithm are quite generic and applicable to different linear, non-linear, and mixed-integer optimization problems. However, according to the *no free lunch* theorem [46], there cannot be a single optimization algorithm which would be best for solving all types of optimization problems. It then makes sense to concentrate on a class of optimization problems (such as linear or quadratic problems or convex programming problems, or multi-modal problems, etc.) and find the best algorithm of solving that particular class of problems.

In order to come up with such an algorithm, one should be open to both classical and evolutionary approaches to optimization and use whatever problem information is available common to that class of problems. The optimization steps and its operators must also be redesigned using the problem information as far as possible. Instead of starting from a random population (which is often done in EAs!), a knowledge-augmented initialization may be better to be used. To illustrate the customization procedure and its efficacy, here, we briefly discuss customizing an EA for solving a particular class of optimization problems – integer linear-programming (ILP) problems, often found in engineering problems.

The integer restriction in ILP problems makes the very efficient real-parameter linear-programming methods to fail. The usual branch-and-bound approach is not computationally very efficient, as for every non-integer value of a variable found in an intermediate LP step, the method suggests branching and solving two different LP problems. Thus, this method has an exponential computational complexity and the computational burden shows up while attempting to solve moderately large-sized problems, such as ILP problems having a few hundreds or thousands of integer variables. In a recent study [19], we have developed a customized EA in which the initialization and genetic operators (crossover and mutation) are custom-made for handling a particular form of linear objective and constraints, often arise in practice:

$$\left.\begin{array}{rll} \text{Maximize} & \frac{1}{H}\sum_{i=1}^{H}\frac{100\sum_{k=1}^{K} w_k x_{ki}}{W_i}, & \\ \text{Subject to} & \sum_{i=1}^{H} x_{ki} = r_k, & \text{for } k = 1,2,\ldots,K, \\ & \sum_{k=1}^{K} w_k x_{ki} \leq W_i, & \text{for } i = 1,2,\ldots,H, \\ & x_{ki} \geq 0, & \\ & x_{ki} \text{ is an integer.} & \end{array}\right\} \quad (3)$$

The above problem comes from a casting scheduling problem, in which certain amount of metal is melt in a crucible in a heat and the task is to decide which castings (of differing weights w_k) are to be made from the heat. There are a total of H heats and K orders and x_{ki} denotes the number of k-th castings which are

to made from the i-th heat. Equality constraints limits the demand of castings (r_k) and the inequality constraints limit the total utilized molten metal by the size (W_i) of the crucible used in the i-th heat. The decision variables are x_{ki} (a matrix of integers) which can easily be a large number for a modest number of demands (K) requiring a modest number of heats (H). In this problem, the true optimum cannot have a function value more than one, thereby providing an upper bound in the objective value.

In solving the above problem using a genetic algorithm with generic operators, not so encouraging results were obtained [19]. This is not surprising because the generic GA operators do not respect (or exploit) the linearity of the problem and therefore end up taking more iterations to come close to the true optimum. However, Figure 4 shows the computational time needed on a Pentium IV, 1 GHz serial PC to solve the above ILP problem having variables from 1,000 to one million using our customized EA. To our knowledge, not many optimization algorithms have been tried to solve such large-sized problems in the past. Interestingly, the figure shows that the computational complexity is less-than quadratic to the number of variables in achieving a solution very close (at most 0.3%) from the true optimum.

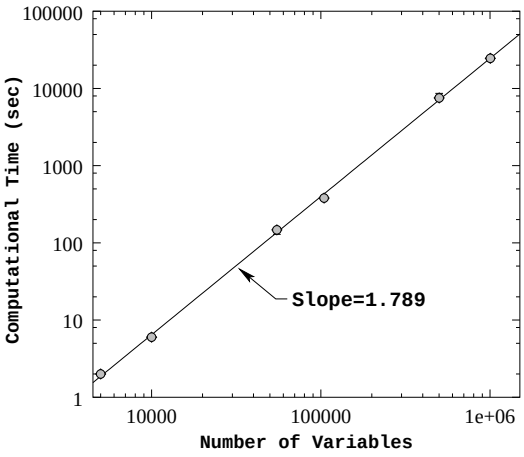

Figure 4: Computational time needed to reach near the optimum of ILP problems having small to large number of variables (*as many as one million!*).

The above study demonstrates two aspects: (i) An EA must be customized for a problem to obtain the real advantage of the population approach and (ii) If done properly, the resulting EA can be found to be *scalable* to a large number of problems. The scalability of optimization algorithms is an important issue and must be ensured for a competent algorithm [29], if it is to be applied on a regular basis.

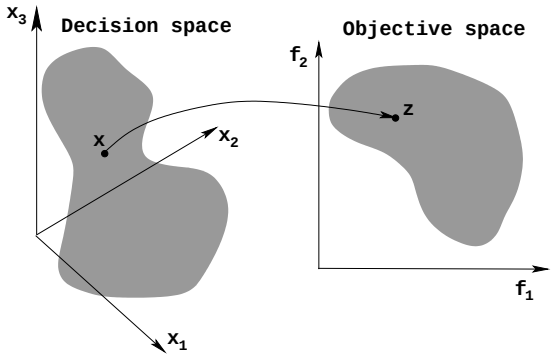

Figure 5: Representation of the decision variable space and the corresponding objective space.

MULTI-OBJECTIVE PROBLEM-SOLVING USING EC

A multi-objective optimization problem involves a number of objective functions which are to be either minimized or maximized. As in the single-objective optimization problem, the multi-objective optimization problem usually has a number of constraints which any feasible solution (including the optimal solution) must satisfy. In the following, we state the multi-objective optimization problem (MOOP) in its general form:

$$\left.\begin{array}{rlll} \text{Minimize/Maximize} & f_m(\mathbf{x}), & & m = 1, 2, \ldots, M; \\ \text{subject to} & g_j(\mathbf{x}) \geq 0, & & j = 1, 2, \ldots, J; \\ & h_k(\mathbf{x}) = 0, & & k = 1, 2, \ldots, K; \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)}, & & i = 1, 2, \ldots, n. \end{array}\right\} \tag{4}$$

A solution $\mathbf{x}$ is a vector of n decision variables: $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$. The solutions satisfying the constraints and variable bounds constitute a *feasible decision variable space* $\mathcal{S}$, or simply the decision space. One of the striking differences between single-objective and multi-objective optimization is that in multi-objective optimization the objective functions constitute a multi-dimensional space, in addition to the usual decision variable space. This additional space is called the *objective space*, $\mathcal{Z}$. For each solution $\mathbf{x}$ in the decision variable space, there exists a point in the objective space, denoted by $\mathbf{f}(\mathbf{x}) = \mathbf{z} = (z_1, z_2, \ldots, z_M)^T$. The mapping takes place between an n-dimensional solution vector and an M-dimensional objective vector. Figure 5 illustrates these two spaces and a mapping between them.

Principles of Multi-Objective Optimization

Most real-world search and optimization problems involve multiple objectives. The extremist principle of finding the optimum solution cannot be applied to one ob-

jective alone, when the rest of the objectives are also important. The presence of multiple conflicting objectives gives rise to a set of trade-off optimal solutions, known as Pareto-optimal solutions [11, 22, 36]. Different Pareto-optimal solutions produce a trade-off (conflicting scenarios) among different objectives. A solution that is better with respect to one objective requires a compromise in at least one other objective. Since many such solutions are targets here, clearly, there are two goals of multi-objective optimization:

1. Find a set of solutions close to the Pareto-optimal solutions, and
2. Find a set of solutions which are diverse enough to represent the entire spread of the Pareto-optimal front.

The Ideal Multi-Objective Optimization

Although the fundamental difference between single and multiple objective optimization lies in the cardinality in the optimal set, from a practical standpoint a user needs only one solution, no matter whether the associated optimization problem is single-objective or multi-objective. In the case of multi-objective optimization, the user is now in a dilemma. Which of these optimal solutions must one choose? This is not an easy question to answer. It involves many higher-level information which are often non-technical, qualitative and experience-driven. However, if a set of many trade-off solutions are already worked out or available, one can evaluate the pros and cons of each of these solutions based on all such non-technical and qualitative, yet still important, considerations and compare them to make a choice. Thus, in a multi-objective optimization, ideally the effort must be made in finding the set of trade-off optimal solutions by considering all objectives to be important. After a set of such trade-off solutions are found, a user can then use higher-level qualitative considerations to make a choice. In view of these discussions, we suggest the following principle for an *ideal multi-objective optimization procedure*:

Step 1 Find multiple trade-off optimal solutions with a wide range of values for objectives.

Step 2 Choose one of the obtained solutions using higher-level information.

Figure 6 shows schematically the principles in an ideal multi-objective optimization procedure. In Step 1 (vertically downwards), multiple trade-off solutions are found. Thereafter, in Step 2 (horizontally, towards the right), higher-level information is used to choose one of the trade-off solutions. With this procedure in mind, it is easy to realize that single-objective optimization is a degenerate case of multi-objective optimization. In the case of single-objective optimization with only one global optimal solution, Step 1 will find only one solution, thereby not requiring us to proceed to Step 2. In the next section, we present an evolutionary multi-objective optimization algorithm to find a set of Pareto-optimal solutions, instead of just one of them.

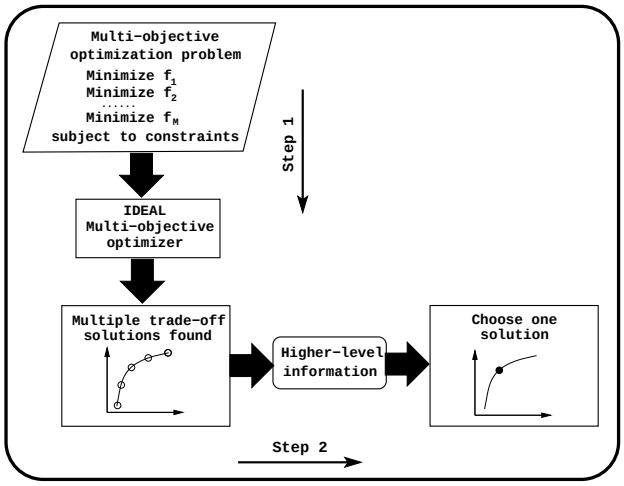

Figure 6: A schematic of an ideal multi-objective optimization procedure.

Elitist Non-Dominated Sorting GA (NSGA-II)

The NSGA-II procedure [15] for finding multiple Pareto-optimal solutions in a multi-objective optimization problem has the following three features:

1. It uses an elitist principle,
2. It uses an explicit diversity preserving mechanism, and
3. It emphasizes non-dominated solutions in a population.

In NSGA-II, the offspring population Q_t is first created by using the parent population P_t and the usual genetic operators [27, 35]. Thereafter, the two populations are combined together to form R_t of size $2N$. Then, a non-dominated sorting is used to classify the entire population R_t. The *domination* between two solutions is defined as follows [11, 36]:

Definition 1 *A solution* $\mathbf{x}^{(1)}$ *is said to dominate the other solution* $\mathbf{x}^{(2)}$*, if both the following conditions are true:*

1. *The solution* $\mathbf{x}^{(1)}$ *is no worse than* $\mathbf{x}^{(2)}$ *in all objectives.*
2. *The solution* $\mathbf{x}^{(1)}$ *is strictly better than* $\mathbf{x}^{(2)}$ *in at least one objective.*

For a given set of solutions (for example, those shown in Figure 7(a)), a pairwise comparison can be made using the above definition and whether one solution dominates the other can be established. All solutions which are not dominated by any other members of the set are called the non-dominated solutions of level one. These solutions (3, 5, and 6 in the figure) are also said to lie on the best non-dominated front. By temporarily discounting these solutions, the above principle

can be followed again and the next non-dominated front (solutions 1 and 4 in the figure) can be identified. Figure 7(b) shows how the given set of six solutions are classified into three non-dominated fronts. Once the non-dominated sorting is over,

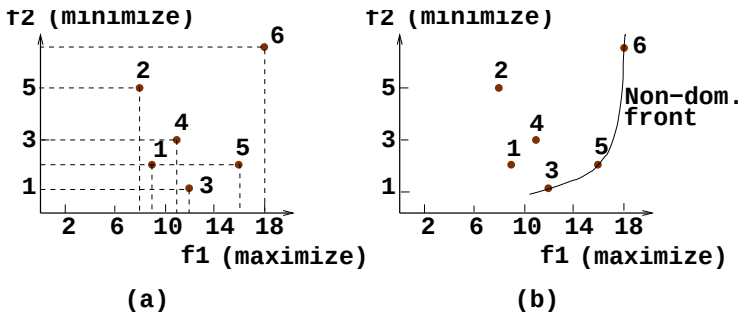

Figure 7: A set of solutions and the best non-dominated front.

the new population is filled by solutions of different non-dominated fronts, one at a time. The filling starts with the best non-dominated front and continues with solutions of the second non-dominated front, followed by the third non-dominated front, and so on. Since the overall population size of R_t is $2N$, not all fronts may be accommodated in N slots available in the new population. All fronts which could not be accommodated are simply deleted. When the last allowed front is being considered, there may exist more solutions in the last front than the remaining slots in the new population. This scenario is illustrated in Figure 8. Instead of arbitrarily discarding some members from the last front, the solutions which will make the diversity of the selected solutions the largest are chosen.

The crowding-sorting of the solutions of front i (the last front which could not be accommodated fully) is performed by using a *crowding distance* metric, denoting the extent of crowding in the objective space with other population members. The population is arranged in descending order of magnitude of the crowding distance values and the required number of solutions are chosen from the top of the list. The crowding distance d_i of a solution i is a measure of the search space around i which is not occupied by any other solution in the population. Here, we simply calculate this quantity d_i by estimating the perimeter of the cuboid formed by using the nearest neighbors as the vertices (we call this the *crowding distance*). In Figure 9, the crowding distance of the i-th solution in its front (marked with filled circles) is the average side-length of the cuboid (shown by a dashed box).

Sample Simulation Results

In this subsection, we show the simulation results of the NSGA-II on two test problems. The first problem (SCH1) is a 30-variable, two-objective problem with a

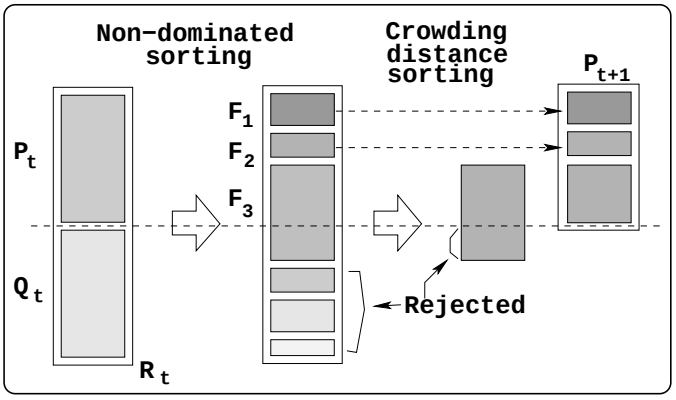

Figure 8: A schematic of the NSGA-II procedure.

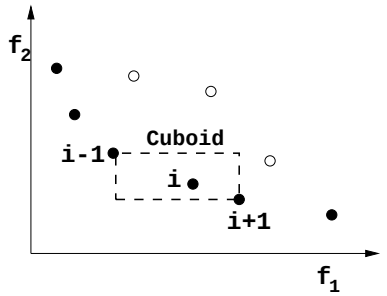

Figure 9: The crowding distance calculation.

convex Pareto-optimal front.

$$\mathsf{ZDT2}: \begin{cases} \text{Minimize} & f_1(x) = x_1, \\ \text{Minimize} & f_2(x) = g\left[1 - (f_1/g)^2\right], \\ \text{where} & g(\mathbf{x}) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \\ & -10^3 \leq x \leq 10^3. \end{cases} \tag{5}$$

The second problem (KUR) has a disjointed set of Pareto-optimal fronts:

$$\mathsf{KUR}: \begin{cases} \text{Minimize} & f_1(\mathbf{x}) = \sum_{i=1}^{2}\left[-10\exp(-0.2\sqrt{x_i^2 + x_{i+1}^2}\right], \\ \text{Minimize} & f_2(\mathbf{x}) = \sum_{i=1}^{3}\left[|x_i|^{0.8} + 5\sin(x_i^3)\right], \\ & -5 \leq x_i \leq 5, \quad i = 1, 2, 3. \end{cases} \tag{6}$$

NSGA-II is run with a population size of 100 and for 250 generations. Figures 10 and 11 show that the NSGA-II converges to the Pareto-optimal front and maintains a good spread of solutions on both test problems.

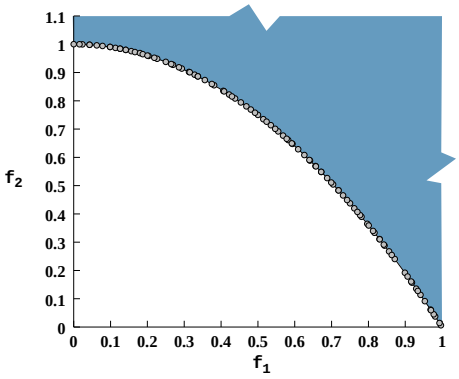

Figure 10: NSGA-II on ZDT2.

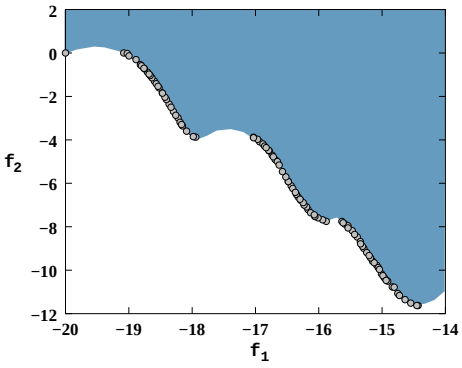

Figure 11: NSGA-II on KUR.

Constraint Handling in Multi-Objective Optimization

The constraint handling method modifies the binary tournament selection of NSGA-II, where two solutions are picked from the population and the better solution is chosen. In the presence of constraints, each solution can be either feasible or infeasible. Thus, there may be at most three situations: (i) both solutions are feasible, (ii) one is feasible and the other is not, and (iii) both are infeasible. We consider each case by simply redefining the domination principle as follows (we call it the *constrain-domination* condition for any two solutions $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$):

Definition 2 *A solution $\mathbf{x}^{(i)}$ is said to 'constrain-dominate' a solution $\mathbf{x}^{(j)}$ (or $\mathbf{x}^{(i)} \preceq_c \mathbf{x}^{(j)}$), if any of the following conditions are true:*

1. *Solution $\mathbf{x}^{(i)}$ is feasible and solution $\mathbf{x}^{(j)}$ is not.*
2. *Solutions $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are both infeasible, but solution $\mathbf{x}^{(i)}$ has a smaller constraint violation.*
3. *Solutions $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are feasible and solution $\mathbf{x}^{(i)}$ dominates solution $\mathbf{x}^{(j)}$ in the usual sense (Definition 1).*

The above change in the definition allows a minimal change in the NSGA-II procedure described earlier. Figure 12 shows the non-dominated fronts on a six-membered population due to the introduction of a constraint. In the absence of the constraint, the non-dominated fronts (shown by dashed lines) would have been `((1,3,5), (2,6), (4))`, but in their presence, the new fronts are `((4,5), (6), (2), (1), (3))`. The first non-dominated front is constituted with the best feasible solutions in the population and any feasible solution lies on a better non-dominated front than an infeasible solution.

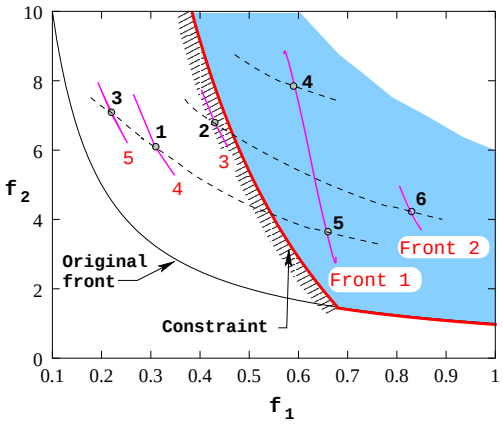

Figure 12: Non-constrain-dominated fronts.

Sample Simulation Results

In the following, we show simulation results of NSGA-II with the above constraint handling mechanism to two test problems (CONSTR (left) and TNK (right)):

$$\begin{aligned} \text{Min.} \quad & f_1(\mathbf{x}) = x_1, \\ \text{Min.} \quad & f_2(\mathbf{x}) = \tfrac{1+x_2}{x_1}, \\ \text{s.t.} \quad & g_1(\mathbf{x}) \equiv x_2 + 9x_1 \geq 6, \\ & g_2(\mathbf{x}) \equiv -x_2 + 9x_1 \geq 1, \\ & 0.1 \leq x_1 \leq 1, \\ & 0 \leq x_2 \leq 5. \end{aligned}$$

$$\begin{aligned} \text{Min.} \quad & f_1(\mathbf{x}) = x_1, \\ \text{Min.} \quad & f_2(\mathbf{x}) = x_2, \\ \text{s.t.} \quad & x_1^2 + x_2^2 - 1 - 0.1\cos\left(16\tan^{-1}\tfrac{x_1}{x_2}\right) \geq 0, \\ & (x_1 - 0.5)^2 + (x_2 - 0.5)^2 \leq 0.5, \\ & 0 \leq x_1 \leq \pi, \\ & 0 \leq x_2 \leq \pi. \end{aligned}$$

With identical parameter settings as before, NSGA-II finds a good distribution of solutions on the Pareto-optimal front in both problems (Figures 13 and 14, respectively).

How is Evolutionary Multi-Objective Optimization Useful?

It is shown above and amply demonstrated in the EMO literature [8] that there exist a number of evolutionary multi-objective optimization (EMO) algorithms for finding a well-converged and a well-distributed set of solutions near the true Pareto-optimal front. In this section, we stress two different purposes for using EMO algorithms in practice:

1. Aid in choosing a compromised solution, and
2. Aid in understanding the optimality properties of solutions better.

We discuss each of the above issues in the following subsections.

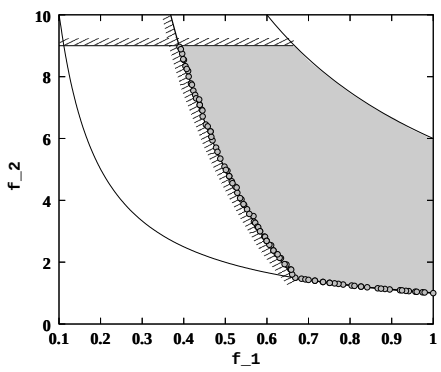

Figure 13: Obtained non-dominated solutions with NSGA-II on the constrained problem CONSTR.

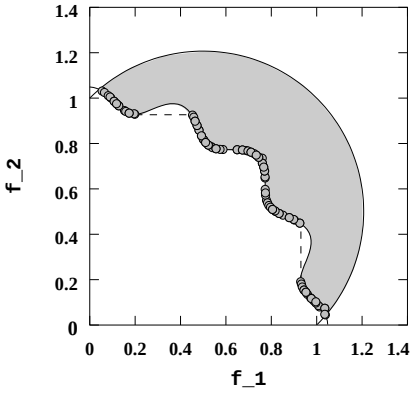

Figure 14: Obtained non-dominated solutions with NSGA-II on the constrained problem TNK.

In Choosing a Compromised Solution

There is no doubt that the availability of a number of compromised solutions is always useful to a decision-maker in choosing a particular solution. Although specific systematic procedures must be outlined to help choose a solution, the task of concentrating in a particular region on the Pareto-optimal front based on some pre-defined weight information or other means and without a prior exploration of various trade-off options may *not* be a desirable procedure. Although a number of subjective considerations can occur in certain applications, here, we discuss a few *post-optimal* generic techniques which can be adopted to concentrate near a particular region on the Pareto frontier.

In the so-called *pseudo-weight method*, the obtained trade-off solutions can be assigned a pseudo-weight based on their positions on the Pareto frontier [11]. After a pseudo-weight is assigned to each solution, the one closer to the decision-maker's wish (or original choice) may be selected. Figure 15 illustrates this procedure.

In the *gain-to-loss ratio method*, the neighboring solutions to each obtained solution may first be identified in each objective direction. Thereafter, the gain in one objective value can be computed for a unit sacrifice in other objectives (or in some of norm of other objectives). When such a gain-to-loss ratio is computed for all obtained trade-off solutions, the solution having the maximum gain-to-loss ratio can be selected. Figure 16 illustrates this procedure.

Finally, a compromise programming technique in which a solution is chosen based on the maximum value of an l_p-norm [36, 11] from an *ideal* point can also be used to select one solution from the set of obtained trade-off solutions. Figure 17 shows the selection solution with $p = 2$ (Euclidean norm).

Although the above metrics are generic and can be applied to any multi-objective

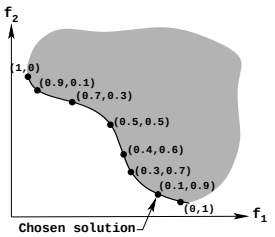

Figure 15: The pseudo-weight method of selecting one trade-off solution.

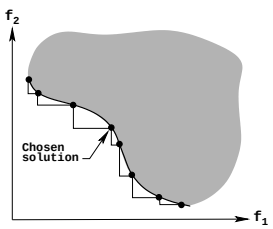

Figure 16: The gain-to-loss ratio method of selecting one trade-off solution.

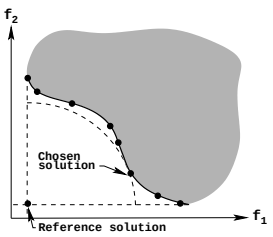

Figure 17: The l_2 method of selecting one trade-off solution.

optimization problem, in practice, a solution will be selected based on some higher-level problem-specific information (often non-technical) by the decision-maker. What we argue here is that the presence of a number of trade-off optimal solutions will make the decision-making easier on the part of the decision-maker. Moreover, since the decision-maker can have an idea of the complete range of the Pareto-optimal front and since the presence of multiple solutions will allow the decision-maker to compare solutions from different regions of the Pareto-optimal front, the decision-maker will make a more *confident* decision by using this procedure.

In Capturing Insights about the Problem

By solving a number of engineering multi-objective optimization problems, a recent study [12] has argued the existence of the following two interesting properties of the Pareto-optimal solutions:

- There exist some *common* properties among Pareto-optimal solutions, and
- There exist some properties which cause a trade-off among the Pareto-optimal solutions.

Although the above observations are purely based on simulation results obtained using the NSGA-II algorithm on a number of case studies, there exist a number of mathematical optimality conditions [11, 36], which every Pareto-optimal solution must satisfy. Putting it otherwise, since all Pareto-optimal solutions have to satisfy some criteria, it may not be surprising that the computer simulations using NSGA-II have discovered such commonality properties among the obtained optimal solutions in different case studies. What we stress here is that if there exist some commonality properties among optimal solutions, they would provide useful information about the problem at hand. Since the mathematical conditions involve gradient computations and solution of a number of non-linear equations, they may be difficult to use to come up with any useful information. The only way to discover such useful information is to first apply an EMO to find a set of real solutions close to the true Pareto-optimal front and then analyze them for discovering any such

properties. We illustrate this principle with the help of two engineering optimization problems.

In the optimal gearbox design problem studied elsewhere [18], the objectives were to minimize the gear-box size and maximize the power rating of the gear-box. The optimization problem involved 29 mixed variables and 101 equality and non-linear inequality constraints. Using the above-described NSGA-II procedure and constraint-handling method, we obtained 300 trade-off solutions for the problem. When we lay the solutions in an increasing order of the power rating and investigate for any pattern in the change in all 29 design variables, the following interesting and useful information were observed:

1. Only one design variables (*module (m)*, a discrete variable) changes with the power rating (p), as shown in Figure 18.

2. Rest all design variables do not change significantly.

The figure shows a monotonic relationship between the two quantities: $m \propto \sqrt{p}$. Such relationships are invaluable to a designer. Let us say that there exist an optimum gear-box (with a gear module m_1) delivering a power p_1. If a new gear-box needs to be designed for four times the power requirement, the above NSGA-II result suggests that the module needs to be increased by a factor of two and all other 28 design variables need not be changed. It is not clear how else to obtain such vital information about a problem rather than finding a set of Pareto-optimal solutions and investigating for any common principles among the solutions.

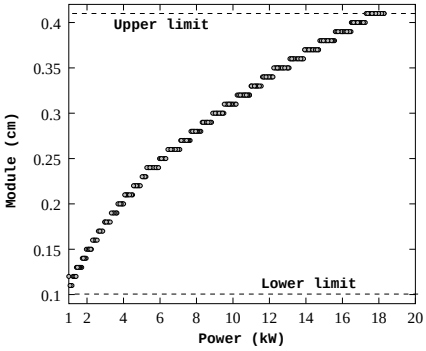

Figure 18: The module is found to follow a monotonic relationship with delivered power in a gear-box design.

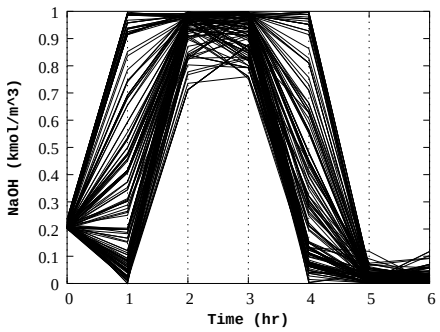

Figure 19: The charge of NaOH has a pattern for optimal production in an epoxy-polymerization process.

To show that such a property can exist in other engineering problems, we present the charging pattern of NaOH in an epoxy-polymerization process for maximizing the average molecular weight of the polymer (Mn) and simultaneously minimizing the polydispersity index (PDI) in Figure 19. NaOH needs to be charged at every hour for six hours along with two other ingredients in such a manner so as to have

a maximum Mn and minimum PDI values of the end-product. The process involves 21 decision variables and two objective functions which require a numerical solution to 48 coupled differential equations. The figure shows the charging pattern of all obtained trade-off solutions. It is interesting to note that although the charging could have been anywhere in the range [NaOH] $\in [0, 1]$ for all seven additions, there is a definite pattern emerging from all solutions. A low-charge to start with, followed by a maximum charge for a while, and then drastically reduce the charge can all be explained from the chemistry of the epoxy-polymerization process, but were not explicitly used or reported in any earlier study. However, such vital operating principles are possible to be emerged from a systematic analysis of finding a set of Pareto-optimal solutions and then investigating them.

CONCLUSIONS

In this paper, we have described the evolutionary optimization methods as a special case of a population-based approach. The strengths of an EA in solving search and optimization problems have been clearly outlined. Of them, the abilities of handling mixed type of variables, handling non-linear constraints, customizing for solving different classes of problems efficiently, and finding multiple trade-off optimal solutions in the presence of multiple conflicting objectives have been mainly discussed. The accompanying case studies from different engineering optimization domains should paint a clear picture in the mind of the readers about the usefulness of EAs in solving complex engineering optimization problems. The use of multiple trade-off solutions to decipher salient insights about a problem has been found to be a hallmark of evolutionary multi-objective optimization. It would be worthwhile if similar such studies are carried out in some important hydroinformatics problems in the near future and similar useful insights are unearthed for the benefit of the researchers and practitioners alike.

REFERENCES

[1] Robert J. Abrahart, Linda See, and Pauline E. Kneale. Using pruning algorithms and genetic algorithms to optimise network architectures and forecasting inputs in a neural network rainfall-runoff model. *Journal of Hydroinformatics*, 1:103–114, 1999.

[2] H.-G. Beyer. *The theory of evolution strategies.* Berlin, Germany: Springer, 2001.

[3] H.-G. Beyer and K. Deb. On the desired behaviour of self-adaptive evolutionary algorithms. In *Parallel Problem Solving from Nature VI (PPSN-VI)*, pages 59–68, 2000.

[4] Donald H. Burn and Jeanne S. Yullanti. Waste-load allocation using genetic algorithms. *Journal of Water Resources Planning and Management*, 127(2):121–129, 2001.

[5] Alcigeimes B. Celeste, Koichi Suzuki, and Akihiro Kadota. Genetic algorithms for real-time operation of multipurpose water resource systems. *Journal of Hydroinformatics*, 6(1):19–38, 2004.

[6] Peter B. Cheung, Luisa F.R. Reis, Klebber T.M. Formiga, Fazal H. Chaudhry, and Waldo G.C. Ticona. Multiobjective evolutionary algorithms applied to the rehabilitation of a water distribution system: A comparative study. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Second International Conference on Evolutionary Multi-Criterion Optimization (EMO-2003)*, pages 662–676, 2003.

[7] Scott E. Cieniawski, J. W. Eheart, and S. Ranjithan. Using genetic algorithms to solve a multiobjective groundwater monitoring problem. *Water Resources Research*, 31(2):399–409, 1995.

[8] C. A. C. Coello. `http://www.lania.mx/~ccoello/EMOO/`, 2003.

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

[10] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2–4):311–338, 2000.

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

[12] K. Deb. Unveiling innovative design principles by means of multiple conflicting objectives. *Engineering Optimization*, 35(5):445–470, 2003.

[13] K. Deb. A population-based algorithm-generator for real-parameter optimization. *Soft Computing*, in press.

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

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

[16] K. Deb, A. Anand, and D. Joshi. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation Journal*, 10(4):371–395, 2002.

[17] K. Deb and M. Goyal. A robust optimization procedure for mechanical component design based on genetic adaptive search. *Transactions of the ASME: Journal of Mechanical Design*, 120(2):162–164, 1998.

[18] K. Deb and S. Jain. Multi-speed gearbox design using multi-objective evolutionary algorithms. *ASME Transactions on Mechanical Design*, 125(3):609–619, 2003.

[19] K. Deb, A. R. Reddy, and G. Singh. Optimal scheduling of casting sequence using genetic algorithms. *Journal of Materials and Manufacturing Processes*, 18(3):409–432, 2003.

[20] Jason L. Dorn and S. Ranji Ranjithan. Evolutionary multiobjective optimization in watershed water quality management. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Second International Conference on Evolutionary Multi-Criterion Optimization (EMO-2003)*, pages 692–706, 2003.

[21] J. W. Eheart, S. E. Cieniawski, and S. Ranjithan. Genetic-algorithm-based design of groundwater quality monitoring system. Technical Report WRC Research Report No. 218, Urbana, IL: Department of Civil Engineering,The University of Illinois at Urbana-Champaign, 1993.

[22] M. Ehrgott. *Multicriteria Optimization.* Berlin: Springer, 2000.

[23] M. Erickson, A. Mayer, and J. Horn. The niched Pareto genetic algorithm 2 applied to the design of groundwater remediation systems. In *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001)*, pages 681–695, 2001.

[24] D. B. Fogel. *Evolutionary Computation.* Piscataway, NY: IEEE Press, 1995.

[25] L. J. Fogel, P. J. Angeline, and D. B. Fogel. An evolutionary programming approach to self-adaptation on finite state machines. In *Proceedings of the Fourth International Conference on Evolutionary Programming*, pages 355–365, 1995.

[26] Klebber T.M. Formiga, Fazal H. Chaufhry, Peter B. Cheung, and Luisa F.R. Reis. Optimal design of water distribution system by multiobjective evolutionary methods. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Second International Conference on Evolutionary Multi-Criterion Optimization (EMO-2003)*, pages 677–691, 2003.

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

[28] D. E. Goldberg. A hydroinformatician's approach to computational innovation and the design of genetic algorithms. *Journal of Hydroinformatics*, 2:155–162, 2000.

[29] D. E. Goldberg. *The design of innovation: Lessons from and for Competent genetic algorithms.* Kluwer Academic Publishers, 2002.

[30] D. Halhal, G. A. Walters, D. Ouazar, and D. A. Savic. Multi-objective improvement of water distribution systems using a structure messy genetic algorithm approach. *ASCE Journal of Water Resources Planning and Management*, 123(3):137–146, 1997.

[31] F. Herrera, M. Lozano, and J. L. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12(4):265–319, 1998.

[32] J. H. Holland. *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: MIT Press, 1975.

[33] Shie-Yui Liong, Soon-Thiam Khu, and Weng Tat Chan. Novel application of genetic algorithm and neural network in water resources: Development of Pareto front. In *Eleventh Congress of the International Association for Hydraulic Research–Asia and Pacific Division*, pages 185–194, Yogyakarta, Indonesia, 1998.

[34] Carlos E. Mariano and Eduardo Morales. A multiple objective ant-q algorithm for the design of water distribution irrigation networks. Technical Report Technical Report HC-9904, Instituto Mexicano de Tecnologa del Agua, 1999.

[35] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Berlin: Springer-Verlag, 1992.

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

[37] J. B. Nixon, G. C. Dandy, and A. R. Simpson. A genetic algorithm for optimizing off-farm irrigation scheduling. *Journal of Hydroinformatics*, 3:11–22, 2001.

[38] S. S. Rao. *Optimization: Theory and Applications.* Wiley, New York, 1984.

[39] Patrick M. Reed, Barbara S. Minsker, and David E. Goldberg. A multiobjective approach to cost effective long-term groundwater monitoring using an elitist nondominated sorted genetic algorithm with historical data. *Journal of Hydroinformatics*, 3(2):71–89, 2001.

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

[41] Brian J. Ritzel, J. Wayland Eheart, and S. Ranjithan. Using genetic algorithms to solve a multiple objective groundwater pollution containment problem. *Water Resources Research*, 30(5):1589–1603, 1994.

[42] C. J. Rouhiainen, M. O. Tade, and G. West. Multi-objective genetic algorithm for optimal scheduling of chlorine dosing in water distribution systems. In C. Maksimovic, D. Butler, and F. Ali Memon, editors, *Proceedings of the International Conference on Computers and Control in the Water Industry (CCWI-2003)*, pages 459–469. Amsterdam: Balkema Publishers, 2003.

[43] H.-P. Schwefel. *Numerical Optimization of Computer Models*. Chichester, UK: Wiley, 1981.

[44] R. Storn and K. Price. Differential evolution – A fast and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[45] V. Rao Vemuri and Walter Cedeo. A new genetic algorithm for multi objective optimization in water resource management. In *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, pages 495–500. Piscataway, New Jersey: IEEE Press, 1995.

[46] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1977.