Genetic Algorithm and Multi-objective Function Optimization with the Jumping Gene (Transposon) Adaptation-A Primer

(Lecture delivered on 14th July 2003)

Prof. SANTOSH K. GUPTA and Dr. SHRIKANT A. BHAT
Department of Chemical Engineering,
Indian Institute of Technology, Kanpur - 208016

Dr. Santosh K. Gupta is a Professor of Chemical Engineering at the Indian Institute of Technology, Kanpur (IIT K). He received his B.Tech. from IIT K in 1968, and his Ph.D, from the University of Pennsylvania, Philadelphia in 1972. He has been a Visiting Professor at the University of Notre Dame, USA (1985-87), the National University of Singapore, Singapore (1998-99), and the University of Wisconsin, Madison, USA (1999 - 2000). He has over 180 research papers, and has authored five text books and a research monograph [Reaction Engineering of Step Growth Polymerization (Plenum, NY, 1987)]. He is a recipient of the 1987 Herdillia Award of the Indian Institute of Chemical Engineers for excellence in fundamental research in Chemical Engineering and was elected Fellow of the Indian Academy of Sciences in 1987. His research interests lie in the areas of simulation of industrial systems, multi-objective optimization (using genetic algorithm and its variants), and on-line optimal control, particularly of 'polymerization' reactors. He has also worked on the modeling of the production of *short* carbon fibers.

The excerpts of the lecture delivered by Prof. Santosh K. Gupta

Prof. Mehrotra, Scientists of National Metallurgical Laboratory and Ladies and Gentlemen, it is a privilege for me to be visiting NML and I thank Prof. S. P. Mehrotra for inviting me to deliver a lecture under the CSIR Diamond Jubilee Lecture Series. I am quite aware of the good R&D work that CSIR and NML have been doing and wish you all success in your celebrations as well as future endeavours.

I am going to deliver a lecture on "Genetic Algorithm and Multi-objective Optimization with the Jumping Gene (Transposon) Adaptation - A Primer". Optimization techniques have long been applied to problems of industrial importance. Several excellent texts[1-5] describe the various methods with examples. These usually involve a single objective function and constraints. Most real-world engineering problems, however, require the simultaneous optimization of *several* objectives (multi-objective optimization) that cannot be compared easily with each other (are non-commensurate), and so cannot be combined into a single, meaningful scalar objective function. An example is the maximization of the product, while minimizing the production of an undesirable side product. A very popular and robust technique for solving optimization problems with a single objective function is genetic algorithm (GA), also referred to as simple GA (SGA). This is a search technique developed by Holland.[6] It mimics the process of natural selection and natural genetics. The Darwinian principle of 'survival of the fittest' is used to obtain the optimal solution. This technique is better than calculus-based methods (both direct and indirect methods) that generally obtain the local optimum, and that may miss the global optimum. This technique does not need derivatives either. A recent adaptation of GA [non dominated sorting genetic algorithm[7] with elitism[8] and the jumping gene operator, NSGA II-JG[9]] has been developed to solve multi-objective function optimization problems. In this paper we describe GA and its adaptations in a manner quite suited to a beginner.

GENETIC ALGORITHM

The optimization problem we have chosen for illustration involves two bounded decision variables, x_1 and x_2, and one constraint, and is given by

$$\begin{aligned}
&\text{Max } I(x_1, x_2) && \equiv \text{ Max } I(\mathbf{x}) \\
&\text{subject to (s.t.)} && : \\
&\text{bounds on } \mathbf{x} && : \ x_i^L \le x_i \le x_i^U ; \ i = 1, 2 \\
&\text{constraint} && : \ f(\mathbf{x}) = 0 \qquad (1)
\end{aligned}$$

We first randomly generate N_p feasible sets of decision variables. The feasible region is shown in Fig. 1 for the above example. However, instead of representing the decision variables in terms of commonly used numbers

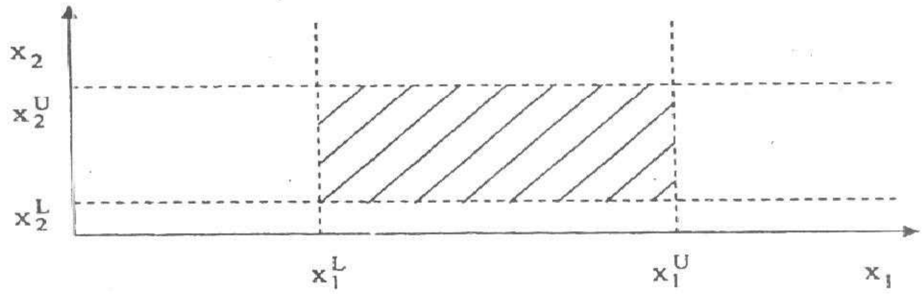

Fig. 1. Feasible region for the 2-decision variable problem in Eq. 1

(in the decimal system), we will use sets of, say, 4 (≡ 1), binary numbers for each decision variable. The set of binaries representing the entire set of two decision variables in this case, is referred to as a 'chromosome' or 'string'. An example of N_p chromosomes generated using a random number generating code (if the random number, R, is less than 0.5, use the binary, 0, while if R is greater than 0.5, use the binary 1) is:

	S_3	S_2	S_1	S_0	S_3	S_2	S_1	S_0	
1st chromosome:	1	0	1	0	0	1	1	1	
2nd chromosome:	1	1	0	1	0	1	0	1	
.									
.									
N_p th chromosome:	1	1	1	1	0	0	0	1	
	substring 1				substring 2				(2)

Here, S_0 ,S_1, S_2, and S_3 denote the binaries in any substring at the 0th, 1st, 2nd and 3rd positions, respectively. The domain, $[\, x_i^L , x_i^U \,]$, for each decision variable is now divided into $(2^l - 1)$[= 15 in the present example] equal intervals. Fig. 2 shows that the lower limit, x_i^U , for a decision variable

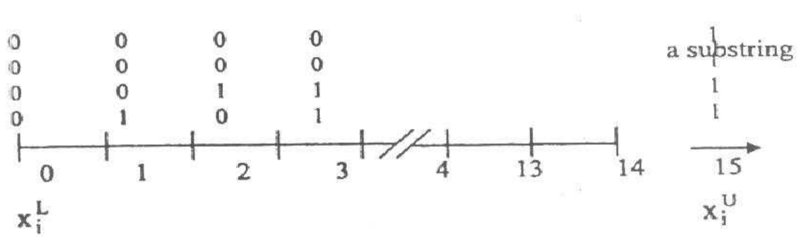

Fig. 2 : Mapping of binary substrings

is assigned to the 'all 0' substring, (0 0 0 0), while the upper limit, x_i^U, to the 'all 1' substring, (1 1 1 1). The other 14 combinations of the substring are sequentially assigned values in-between the bounds of x_i, as shown in Fig. 2. The binary substrings are mapped into real numbers using the following mapping rule:

$$x_i = x_i^L + \frac{x_i^U - x_i^L}{2^l - 1} \times \left(\sum_{i=0}^{l-1} 2^i S_i \right) \tag{3}$$

Clearly, the longer the substring, the larger are the number of intervals in $[x_i^L, x_i^U]$, and the higher the accuracy of search. The mapped real values of each of the (two, here) decision variables, can be used with a model to evaluate the value of the objective function, $I_j(\mathbf{x}_j)$, for the jth chromosome. I_j are referred to as fitness values. For the minimization of $I(\mathbf{x})$, the fitness function (to be maximized) can be taken as $1/(1 + I_j)$.

At this stage, we have a set of N_p feasible solutions (parent chromosomes), each associated with a fitness value. We now need to generate improved chromosomes (daughters) using an appropriate methodology. This is done by mimicking natural genetics. We make N_p copies of the parent chromosomes at a new location, called the 'gene pool', using proportionate representation based on the 'goodness' of the chromosome, i.e., the better the jth chromosome (in terms of I_j), the higher is its chance of getting copied. The probability, P_j, of selecting the jth string for copying is taken as:

$$P_j = \frac{I_j}{\sum_{i=1}^{N_p} I_i} \; ; \qquad j = 1, 2, \ldots, N_p \tag{4}$$

The actual methodology used for this is the roulette wheel selection. We divide the range, $0 \le R \le 1$, of random number, R, into N_p zones: $0 \le R \le P_1$; $P_1 \le R \le P_1 + P_2$; ....;

$\sum_{j=1}^{N_p - 1} P_j \le R \le \sum_{j=1}^{N_p} P_j = 1$ and assign chromosomes, 1, 2, . . . , N_p, to these

zones, respectively. For example, if we have five randomly generated chromosomes with the characteristics shown in Table 1, we can partition the range, [0, 1], of R and associate the appropriate chromosome

Table 1 : Five chromosomes with their fitness values

Chromosome Number	I_j	$P_j = I_j / \Sigma I_j$	Range
1	25	0.25	$0 \leq R \leq 0.25$
2	5	0.05	$0.25 \leq R \leq 0.3$
3	40	0.40	$0.3 \leq R \leq 0.7$
4	10	0.10	$0.7 \leq R \leq 0.8$
5	20	0.20	$0.8 \leq R \leq 1$

to each zone. A random number, A, with $0 \leq A \leq 1$, is now generated and the corresponding chromosome is copied into the 'gene pool'. For example, if A is obtained as 0.45, string 3 is copied. This procedure is repeated N_p times. Clearly, chromosomes having a higher fitness will be selected more frequently in the gene pool. Due to the randomness associated with the copying procedure, there are chances that some 'poor' chromosomes get copied (survive). This helps maintain diversity of the gene pool (two 'idiots' can produce a genius, etc.).

Crossover operation is now performed on the chromosomes of the gene pool. We first select any two strings in the gene pool, randomly. The chromosomes in the gene pool are assigned a number, from 1 to N_p. The first random number, C $(0 \leq C \leq 1)$, is generated. This is multiplied by N_p and rounded off into an integer. The chromosome in the gene pool corresponding to this integer is selected. A second chromosome is then selected. This procedure is repeated to give $N_p/2$ pairs. We check if we need to carry out crossover on a pair, using the crossover probability, P_c. A random number in the range [0, 1] is generated for a selected pair. Crossover is performed if this number happens to lie between 0 to P_c. If the random number lies in $[P_c, 1]$, we copy the pair without carrying out crossover. Thus, $100(1 - P_c)$ % of strings remain unchanged in the next generation. This helps in preserving some of the *elite* members of the parent population in the next generation.

Crossover involves selection of a location (crossover site) in the string, randomly, and swapping the two strings at this site, as shown in next page :

$$\begin{array}{ccc} 1001\ 1 \mid 100 & & 1001\ 1 \mid 101 \\ 1011\ 0 \mid 101 & \Rightarrow & 1011\ 0 \mid 100 \\ \text{parent chromosomes} & & \text{daughter chromosomes} \end{array} \qquad (5)$$

In the above pair, there are seven possible internal crossover sites. Generating a random number, $B (0 \le B \le 1)$, and comparing it with zones $[0 \le B \le 1/7\ ; 1/7 \le B \le 2/7\ ; \ldots ; 6/7 \le B \le]$ in an equi-partitioned roulette wheel helps decide the location of the crossover in this pair.

If we somehow have a population in which all chromosomes happen to have a 0 in, say, the first location, it will be impossible to get 1 at this position using crossover. If the optimal solution has 1 at the first position, then we will not be able to obtain it. Similar is the problem at all locations. This drawback is overcome through the mutation operation that follows crossover. In this, all the individual binaries in *all* N_p chromosomes are checked and changed from 0 to 1 (or *vice-versa*) with a small mutation probability, P_m, i.e., if the random number generated corresponding to any binary lies between 0 to P_m, mutation is performed. Too large a value of P_m leads to oscillations of the solutions.

The crossover and mutation operations may create inferior strings but we expect them to be eliminated over successive generations by the copying operation (survival of the fittest). Since SGA works probabilistically with several solutions simultaneously, we can get multiple optimal solutions, if present. For the same reason, SGA does not get stuck in the vicinity of any local optimal solution, and so is a very robust algorithm.

A constraint in the optimization problem can be taken care of by adding it to the objective function in the form of a penalty function.[10-12] The penalty function reduces (for a maximization problem) the modified fitness value of the objective function by assigning a heavy penalty in case the constraint is violated, thus favoring its elimination over the following generations.

We now solve the following constrained optimization problem:

$$\text{Max } I(x_1, x_2) = \frac{1}{1 + [(x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2]} \qquad \text{(a)}$$

s.t.: $(x_1 - 5)^2 + x_2^2 - 26 \geq 0$ (b)

bounds: $0 \leq x_1 \leq 5;\ 0 \leq x_2 \leq 5$ (c) (6)

The modified objective function can be written as:

$$\text{Max } I(x_1, x_2) = \frac{1}{1 + [(x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2]} - w_1[(x_1 - 5)^2 + x_2^2 - 26]^2 \quad (7)$$

Here, w_1 is taken to be a large positive number (depending on the value of the original objective function) in case the constraint is violated, else it is assigned a value of zero. The results for this problem for the 40th generation are shown in Fig. 3. The computational parameters used are: $l = 10$, $P_c = 0.85$, $P_m = 0.01$, $N_p = 100$, $w_1 = 10^3$. Fig. 3 shows that most of the solutions crowd around the optimal point, $(0.829, 2.933)^T$. There are still a few points that violate the constraint.

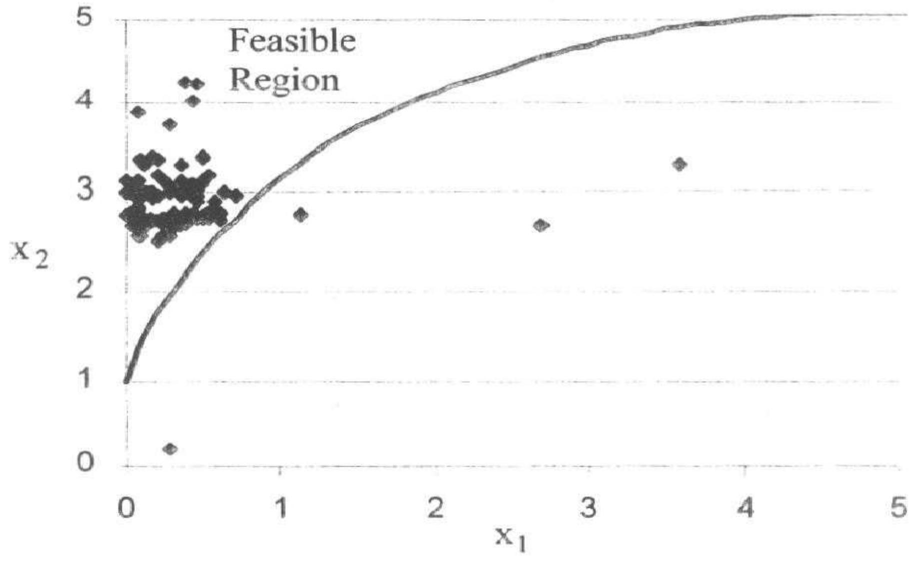

Fig. 3 : Population at the 40th generation for the constrained optimization problem in Eq. 6

With this basic background we now turn our attention to the multiobjective adaptation of GA, namely, NSGA II-JG.[9]

NSGA II-JG[9]

As soon as we have optimization problems involving two or more objective functions (with constraints and bounds), the scene changes and we *may* not get a single, unique optimal solution. Indeed, in several such problems, a set of equally good (non-dominating) optimal solutions, called a Pareto set, is obtained. A typical Pareto set is shown[12, 13] in Fig. 4. Fig. 4a corresponds to the minimization of both the objective functions,

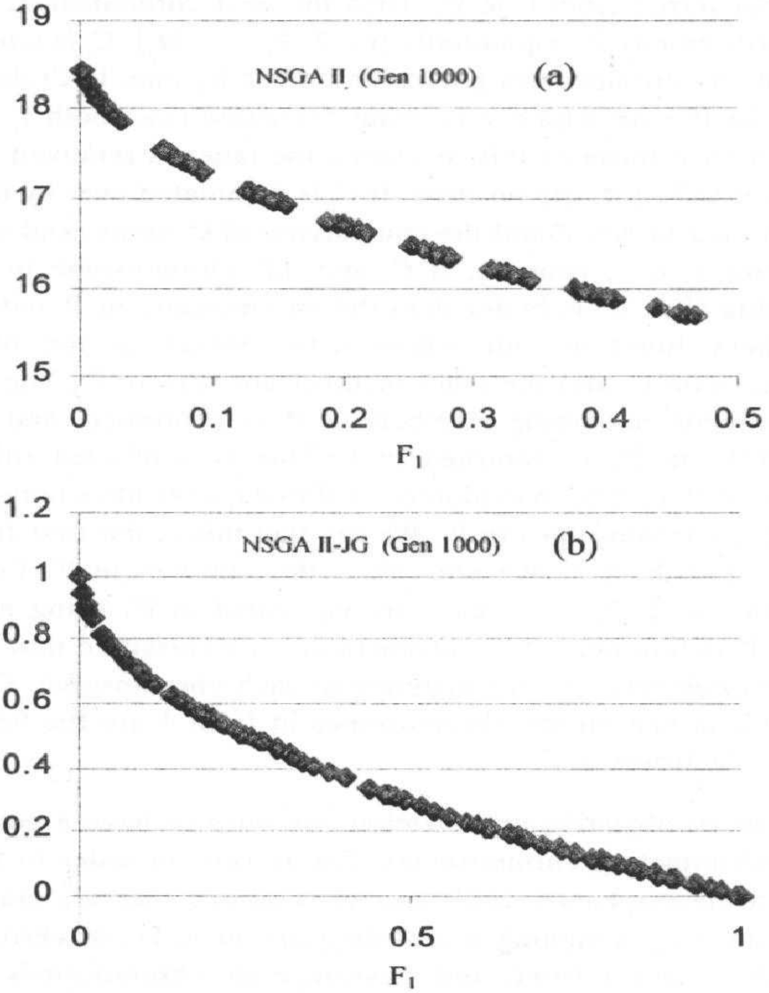

Fig. 4 : The Pareto set obtained for the ZDT4 problem[13] using NSGA II[8] and NSGA II-JG[9]

F_1 and F_2. It is clear that if we take any two points on the curve, one is superior in terms of one objective function, but inferior in terms of the other objective function. SGA has to be adapted so as to be able to solve such multi-objective problems. A popular technique is the non-dominated sorting genetic algorithm, NSGA II,[8, 12] which is described below.

We generate, randomly, N_p parent chromosomes (as in SGA), in box P. The binary substrings are mapped and the model is used to evaluate both $I_1(\mathbf{x})$ and $I_2(\mathbf{x})$, the two objective functions to be maximized. We create a new box, P', having N_p locations. Chromosome 1 (referred to as C_1) is transferred from P to P'. Then the next chromosome, C_i, is taken temporarily to box P' sequentially [$i = 2, 3, \ldots, N_p$]. C_i is compared with all the other chromosomes present in P', one by one. If Ci dominates the member of P' with which it is being compared (i.e., both I_1 and I_2 of C_i are better than those of this member), the latter is removed from P' and put back into P at its old position. If C_i is dominated over by this member, C_i is returned to box P and the comparison of C_i stops (and we study the next member, C_{i+1}, from P). If C_i and this chromosome in P' are non-dominating (i.e., C_i is better than the chromosome in P' with respect to one fitness function, but worse with respect to the other fitness function), both C_i and the other member are kept in P'. The comparison of C_i with the remaining members of P' is continued, and either C_i is finally kept in P', or returned to P. This is continued till all the N_p members in P have been explored. At this stage we have only ($\leq N_p$) non-dominated chromosomes in P'. We say that this is the first front, and we assign it a rank of 1. We now close this sub-box in P'. Further fronts (with ranks = 2, 3, . . . , etc.) are generated in P', using members left in P, till P' is full, i.e., all N_p members in P are classified into fronts. Rank (or front) numbers, R_i, are assigned to each chromosome, C_i, in box P'. It is obvious that all the chromosomes in front 1 are the best, followed by those in fronts 2, 3, . . .

In multi-objective optimization, we wish to have a good spread of the non-dominating chromosomes (Pareto set). In order to achieve this, we try to de-emphasize (kill slowly) solutions that are closely spaced. This is done by assigning a *crowding distance*, D_i, to chromosome, C_i, in P'. We select a front, and re-arrange its chromosomes in order of increasing values of I_1 (or I_2). Then we find the largest cuboid enclosing chromosome, C_i, in the front, that just touches its nearest neighbours. The crowding distance for C_i is calculated as:

$$D_i = \frac{1}{2}[\text{sum of all the sides of the cuboid}] \qquad (8)$$

The lower the value of D_i, the more crowded is the chromosome, C_i. Boundary chromosomes are assigned high values (arbitrarily) of D_i, so as to prevent their killing. Clearly, if we look at two chromosomes, i and j, in P', C_i is better than C_j if $R_i < R_j$. If, however, $R_i = R_j$, then C_i is better than C_j if $D_i > D_j$.

Now that we know how to compare chromosomes in P', we start to copy them in a gene pool (box P"). We take any two members from P' *randomly* (as described for SGA), and make a copy of the better chromosome in P". We then put both the chromosomes back into P'. This is repeated till P" has Np members. Crossover and mutation are now carried out on the chromosomes in the gene pool, as in SGA, giving N_p daughters in box D.

The jumping gene (JG) operation is now performed. Before we describe this operation in NSGA II-JG, we give the salient features of JG. In biology, a jumping gene or transposon is a DNA that can jump in and out of chromosomes. These genes are found to generate genetic variation (diversity) in natural populations. Transposons are of several different kinds. We focus attention on only one kind, namely, those involved in the process known as insertion. Transposons, having a relatively small size of about 1-2 kb (kilo-bases or kilo-nucleotides), can get *inserted* (*replace* a sequence of bases) in a chromosome. These are referred to as insertion sequences. These consist of a central coding sequence of bases that is flanked on both sides by short, inverted, repeat sequences (see Fig. 5). We assume in the JG adaptation of NSGA II that the length (number of base pairs) of the JG is the same as that of the replaced bases in the original chromosome. This keeps the total length of the chromosome unaffected. Only a fraction, P_{jump}, of chromosomes

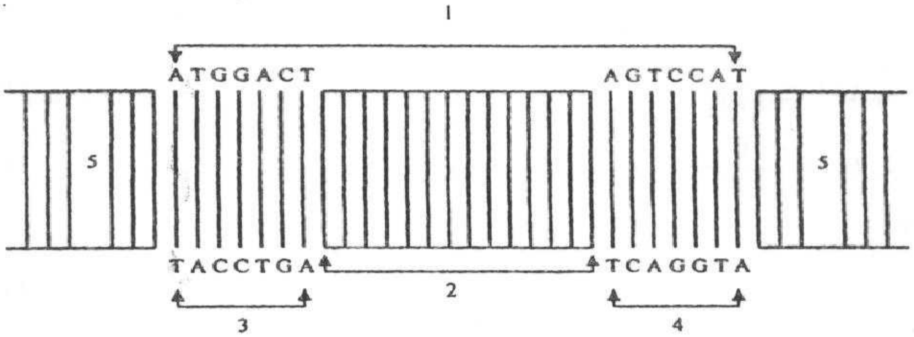

Fig. 5 : Schematic representation of a transposon. 1: Transposon inserted in a chromosome; 2: Genes in the transposon; 3,4: Inverted repeat sequences of bases/nucleotides; 5: Double-stranded DNA of original chromosome. Bases: A: Adenine, C: Cytosine, G: Guanine, T: Thymine.

(selected randomly) in the daughter population (D), are modified by the JG operator. Once a chromosome is identified (using P_{jump}) for the JG operation, two sites are found between which replacement is to occur. This is done using random numbers. We generate a random number (between 0 and 1) and multiply this by the total number of binaries in a chromosome. We round-off to convert it into an integer. This represents the position of one end (beginning) of a transposon. This step is repeated to get the other end of the JG. The binaries between these locations are deleted and replaced by an equal number of newly generated binaries (the procedure is the same as used in SGA) using a sequence of random numbers. We assume only a single transposon to be present in any selected chromosome to keep the algorithm simple (several other possibilities mimicking transposons in natural genetics can be tried). It is worth mentioning at this stage that replacement by a JG involves a macro-macro-mutation that is expected to provide higher genetic diversity.

All the parents in P'' and the N_p daughters with transposons are copied in box, PD (of size $2N_p$). These $2N_p$ chromosomes are re-classified into fronts (kept in box PD') using *only* the criterion of non-domination (and not crowding). The best Np chromosomes are now selected from box PD' and put into box P'''. Clearly, the elite members of the parent population will also get represented in P'''. This completes one generation. This procedure is repeated for the next generation, starting from P''' as the new parent population. Unfortunately, the diversity decreases because of elitism, but this is counter-acted by the JG operation.

We now illustrate NSGA II-JG using the following, two-objective function problem (ZDT4[13]):

$$\text{Max } I_1(\mathbf{x}) = \frac{1}{1+F_1} \qquad \text{(a)}$$

$$\text{Max } I_2(\mathbf{x}) = \frac{1}{1+F_2} \qquad \text{(b)}$$

where,

$$F_1 = x_1 \qquad \text{(c)}$$

$$F_2 = 1 - \left\{ \frac{F_1}{\left[91 + \sum_{i=2}^{10} \left[x_i^2 - 10\cos(4\pi x_i) \right] \right]} \right\}^{1/2} \qquad \text{(d)} \qquad (9)$$

This problem has 21^9 Pareto sets[12,13] and the global Pareto has $0 \leq x_1 \leq 1$ and $x_j = 0$ for j = 2, 3, . . . , 10 (and so, $0 \leq F_1, F_2 \leq 1$). The Pareto results[9] using both NSGA II[8] and NSGA II-JG,[9] are shown in Fig. 4. For this problem, the binary-coded NSGA-II has been found to converge to different *local* Paretos (depending on the computational parameters used), rather than to the global optimal set (note that the optimal values of F_2 in Fig. 4 for NSGA-II are not in the range of 0 - 1, characteristic of the global Pareto). However, the results obtained using NSGA II-JG outperform those obtained by NSGA-II and the global Pareto is, indeed obtained.

It may be mentioned here that NSGA II-JG has also been applied to an industrial fluidized-bed catalytic cracking unit (FCCU).[9] It is found that the Pareto optimal solutions are obtained in only 10 generations, as against 50 required for NSGA II. A more recent adaptation of the JG operator has been used[14] to optimize froth-flotation circuits in the mineral processing area. In this, some substrings (representing the fraction of flow rates of exit streams from a cell to another cell) are *randomly* assigned either 'all 0' (no flow) or 'all 1' (entire stream directed to one unit) binaries during the JG operation. This artificially increases the probability of deleting flows in circuits, something that is difficult to achieve by the normal, unbiased, generation of binaries.

Several industrial processes in chemical engineering have been optimized by our group in the last few years. These are summarized in Table 2, and give a flavor of the immense possibilities still to be tapped.

Table 2. Multi-objective optimizations in chemical engineering studied by our group

Problem Studied	*Reference*
Dynamic optimization of a non-vaporizing nylon 6 batch reactor	Wajge and Gupta[15]
Optimization of an industrial semi-batch nylon 6 batch reactor	Sareen and Gupta[16]
Dynamic optimization of an industrial nylon 6 semi-batch reactor	Mitra et al.[17]
Free radical bulk polymerization reactor	Garg and Gupta[18]

Contd.

Problem Studied	*Reference*
Industrial nylon 6 semi-batch reactor system	Gupta and Gupta[19]
Industrial wiped film polyethylene terephthalate (PET) finishing reactor	Bhaskar et al.[20]
Polymethyl methacrylate (PMMA) reactors and film production	Zhou et al.[21]
Steam reformers	Rajesh et al.[22]
Cyclone separators	Ravi et al.[23]
Venturi scrubbers	Ravi et al.[24]
Beer dialysis	Yuen et al.[25]
Fluidised-bed catalytic cracking unit	Kasat et al.[26]

CONCLUSION

An adaptation of SGA, namely, NSGA II using the jumping gene operator, is described. This algorithm is effective in obtaining global Pareto solutions for problems with multiple local Paretos (where NSGA II fails). This adaptation also converges to optimal solutions faster, as compared to NSGA II. This can prove quite valuable for solving similar compute-intense multi-objective optimization schemes in metallurgical operations.

ACKNOWLEDGEMENT

Partial financial support from DST, GOI, New Delhi [through grant III-5(13)/2001-ET] is grateful acknowledged.

REFERENCES

1. Beveridge, G. S. G. and Schechter, R. S. Optimization: Theory and Practice. McGraw Hill, New York, 1970.

2. Bryson, A. E. and Ho, Y. C. Applied Optimal Control. Blaisdell, Waltham, MA, 1969.

3. Edgar, T. F. and Himmelblau, D. M. Optimization of Chemical Processes. McGraw Hill, New York, 1988.

4. Lapidus, L. and Luus, R. Optimal Control of Engineering Processes. Blaisdell, Waltham, MA, 1967.

5. Ray, W. H. and Szekely, J. Process Optimization with Applications in Metallurgy and Chemical Engineering. Wiley, New York, 1973.

6. Holland, J. H. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI, 1975.

7. Srinivas, N. and Deb, K. Multi-objective Function Optimization using Non-dominated Sorting Genetic Algorithms. Evolutionary. Computation, 2, 221, 1995.

8. Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. A Fast and Elitist Multi-objective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6, 182, 2000.

9. Kasat, R. B. and Gupta, S. K. Multi-objective Optimization of an Industrial Fludized-bed Catalytic Cracking Unit (FCCU) using Genetic Algorithm (GA) with the Jumping Gene Operator. Computers and Chemical Engineering, 00, 000, 2003.

10. Goldberg, D. E. Genetic Algorithms in Search, Optimization and Machine Learning. Addision-Wesley, Reading, MA, 1989.

11. Deb, K. Optimization for Engineering Design: Algorithms and Examples. Prentice Hall of India, New Delhi, 1995.

12. Deb, K. Multi-objective Optimization using Evolutionary Algorithms. Wiley, Chichester, UK, 2001.

13. Zitzler, E., Deb, K. and Thiele, L. Comparison of Multi-objective Evolutionary Algorithms: Empirical Results. Evolutionary Computation, 8, 125, 2000.

14. Guria, C., Verma, M., Mehrotra, S. P. and Gupta, S. K. Multi-objective Optimization of Froth Flotation Circuits using the Elitist, Jumping Gene Adaptation of Genetic Algorithm (GA), in preparation.

15. Wajge, R. M. and Gupta, S. K. Multi-objective Dynamic Optimization of a Non-vaporizing Nylon 6 Batch Reactor. Polym. Eng. Sci., 34, 1161, 1994.

16. Sareen, R. and Gupta, S. K. Multi-objective Optimization of an Industrial Semi-batch Nylon 6 Reactor. J. Appl. Polym. Sci., 58, 2357, 1995.

17. Mitra, K., Deb, K. and Gupta, S. K. Multi-objective Dynamic Optimization of an Industrial Nylon 6 Semi-batch Reactor using Genetic Algorithm. J. Appl. Polym. Sci., 69, 69, 1998.

18. Garg, S. and Gupta, S. K. Multi-objective Optimization of a Free Radical Bulk Polymerization Reactor using Genetic Algorithm. Macromol. Theor. Simul., 8, 46, 1999.

19. Gupta, R. R., and Gupta, S. K. Multi-objective Optimization of an Industrial

Nylon 6 Semi-batch Reactor System using Genetic Algorithm. J. Appl. Polym. Sci., 73, 729, 1999.

20. Bhaskar, V., Gupta, S. K. and Ray, A. K. Multi-objective Optimization of an Industrial Wiped Film Poly (ethylene terephthalate) Reactor, AIChE J., 46, 1046, 2000.

21. Zhou, F., Gupta, S. K. and Ray, A. K. Multi-objective Optimization of the Continuous Casting Process for Poly (methyl methacrylate) using Adapted GA, J. Appl. Polym. Sci., 78, 1439, 2000.

22. Rajesh, J. K., Gupta, S. K., Rangaiah, G. P. and Ray, A. K. Multi-objective Optimization of Industrial Hydrogen Plants, Chem. Eng. Sci., 56, 999, 2001.

23. Ravi, G., Gupta, S. K. and Ray, M. B. Multi-objective Optimization of Cyclone Separators, Ind. Eng. Chem. Res., 39, 4272, 2000.

24. Ravi, G., Gupta, S. K., Viswanathan, S. and Ray, M. B. Optimization of Venturi Scrubbers Using Genetic Algorithm, Ind. Eng. Chem. Res., 41, 2988, 2002.

25. Yuen, C. C., Aatmeeyata, Gupta, S. K. and Ray, A. K. Multi-objective Optimization of Membrane Separation Modules Using Genetic Algorithm, J. Membrane Sci., 176, 177, 2000.

26. Kasat, R. B., Kunzru, D., Saraf, D. N. and Gupta, S. K. Multi-objective Optimization of Industrial FCC Units using Elitist Non-Dominated Sorting Genetic Algorithm, Ind. Eng. Chem. Res., 41, 4765, 2002.