Supporting Text

Materials and methods

The multimethod evolutionary optimization algorithm called **A M**ulti-**AL**gorithm, **G**enetically **A**daptive **M**ultiobjective optimization method, or AMALGAM, outlined in the paper, builds on two new concepts, multimethod search, and adaptive offspring creation, to ensure a fast, reliable and computationally efficient solution to multiobjective optimization problems. A flowchart with algorithmic steps is given in Fig. 4. Our implementation of the method uses four commonly used algorithms: the Nondominated Sorting Genetic Algorithm (1), Particle Swarm Optimization (2), Adaptive Metropolis Search (3), and Differential Evolution (4) to find a well-distributed set of Pareto solutions within a single optimization run. In the following we discuss why the individual algorithms were selected, how they create offspring, and discuss the values of the algorithmic parameters.

Nondominated Sorting Genetic Algorithm II (NSGA-II). The NSGA-II algorithm developed by Deb *et al.* (1) has received the most attention of all evolutionary multiobjective algorithms because of its simplicity and demonstrated superiority over other existing methods. The algorithm uses the well-known genetic operators of selection, crossover, and mutation to create a new population of points Q_{t+1} from an existing population, P_{t+1}. We have implemented the latest version of the code as presented on the website: *http://www.iitk.ac.in/kangal/codes.shtml*, and use the simulated binary crossover (SBX) operator and polynomial mutation (5) to create offspring. In all of our calculations, the crossover and mutation probability were set to 0.9 and $1/n$, respectively (where n denotes the number of parameters), whereas the values of the distribution indices for crossover and mutation operators were set at 20. These values of the algorithmic parameters are identical to those used in ref. 1.

Particle Swarm Optimization (PSO). Particle swarm optimization (PSO) is a population-based stochastic optimization method whose development was inspired from the flocking and swarm behavior of birds and insects. After its introduction in 1995 (6), the method has gained rapid popularity in many fields of study. The method works with a group of potential solutions, called particles, and searches for optimal solutions by continuously modifying this population in subsequent generations. To start, the particles are assigned a random location and velocity in the *n*-dimensional search space. After initialization, each particle iteratively adjusts its position according to its own flying experience, and according to the flying experience of all other particles, making use of the best position encountered by itself, $\mathbf{x}_{\text{best}}$ and the entire population, $\mathbf{p}_{\text{best}}$. In contrast to the NSGA-II algorithm, PSO combines principles from local and global search to evolve a population of points toward the Pareto-optimal front.

The reproductive operator for creating offspring from an existing population is (2,7):

$$\mathbf{v}_{t+1}^{i} = w \cdot \mathbf{v}_{t}^{i} + c_1 r_1 (\mathbf{x}_{\text{best}}^{i} - \mathbf{x}_{t}^{i}) + c_2 r_2 (\mathbf{p}_{\text{best}} - \mathbf{x}_{t}^{i})$$
$$\mathbf{x}_{t+1}^{i} = \mathbf{x}_{t}^{i} + \mathbf{v}_{t+1}^{i} \qquad (1)$$

where $\mathbf{v}_{t}^{i}$ and $\mathbf{x}_{t}^{i}$ represent the current velocity and location of a particle, *w* is the inertia factor, c_1 and c_2 are weights reflecting the cognitive and social factors of the particle, respectively, and r_1 and r_2 are uniform random numbers between 0 and 1. Based on recommendations in previous work (7), the values for c_1 and c_2 were set to 1.5, and the inertia weight was computed as: $w = (1/2) + (1/2)u$, where *u* is a uniform random number between 0 and 1. To be able to escape from local optimal solutions, a turbulence factor is added to the position of each of the individual particles (8):

$$\mathbf{x}_{t+1}^{i} = \mathbf{x}_{t}^{i} + R_T \mathbf{x}_{t}^{i} \quad (2)$$

where $R_T \in [-1,1]$ is a uniform random variable.

An important issue that deserves further attention is how to derive $\mathbf{x}_{best}$ and $\mathbf{p}_{best}$ in our multiobjective optimization. It seems natural to consider an approach in which the quality of the solutions is judged based on the distance of their objective function values to the Pareto-optimal solution set. However, because the location of the Pareto set is unknown, we instead compute the Euclidean distance of all individual solutions to the best values of the objective functions found so far, and store this information in *D*. After this, we sort *D* in order of increasing Euclidean distance, and assign the parameter values corresponding to $D(1)$ to $\mathbf{p}_{best}$ and $D(1\ldots N)$ to $\mathbf{x}_{best}$, where N denotes the population size.

Adaptive Metropolis Search (AMS). The evolutionary algorithms discussed above employ a population-based search procedure to conduct an efficient search of the parameter space. Despite their usefulness and strength, many if not all of these methods exhibit genetic drift in which the majority of the population is inclined to converge toward a single solution, thereby relinquishing occupations in other parts of the search space. Adaptive Metropolis Search (AMS) is a Markov Chain Monte Carlo (MCMC) sampler that actively prevents the search of becoming mired in the relatively small region of a single best solution by adopting an evolutionary strategy that allows replacing parents with offspring of lower fitness (9). While this is a much appreciated strength, the AMS algorithm has another desirable property that is of more interest in the current study: the sampler is very efficient in sampling from high-dimensional distributions. So, if our multimethod evolutionary optimization has progressed toward the Pareto-optimal front, then the AMS algorithm is able to rapidly explore the entire Pareto distribution, successively visiting and generating a large number of solutions.

To create offspring, we implement the following reproductive operator:

$$\mathbf{x}_{t+1}^{i} = N(\mathbf{x}_{t}^{i}, c_n^2 \sum_t) \quad (3)$$

where $\mathbf{x}_t^i$ and Σ_t represent the current location and covariance, respectively, of the best nondominated set of solutions in population P_t. The jumprate parameter c_n directly

determines the spread of the solutions around $\mathbf{x}_t^i$. As a basic choice, the value of c_n was set to $2.4/\sqrt{n}$ (10), where n represents the number of decision variables / parameters.

Differential Evolution (DE). While traditional evolutionary algorithms are well suited to solve many difficult optimization problems, interactions among decision variables (parameters) introduces another level of difficulty in the evolution. Previous work has demonstrated the poor performance of a number of multiobjective evolutionary optimization algorithms, including the NSGA-II, in finding Pareto solutions for rotated problems exhibiting strong interdependencies between parameters (1). Rotated problems typically require correlated, self-adapting mutation step sizes to make timely progress in the optimization.

Differential evolution (DE) has been demonstrated to be able to cope with strong correlation among decision variables, and exhibits rotationally invariant behavior (4). DE is a population-based search algorithm that iteratively modifies an initial population of points in subsequent generations. The method differs from other evolutionary algorithms in the mutation and recombination phase. Unlike Genetic Algorithms and other evolutionary strategies, DE uses weighted differences between solution vectors to create new offspring solutions from the existing population:

$$\mathbf{x}_{t+1}^i = \mathbf{x}_t^i + K(\mathbf{x}_t^a - \mathbf{x}_t^i) + F(\mathbf{x}_t^b - \mathbf{x}_t^c) \quad (4)$$

where K and F are scaling parameters controlling the level of combination between individual solutions, and a, b, and c are randomly selected numbers from $\{1,2,\ldots,N\}$; $a \neq b \neq c \neq i$. The values of K and F were randomly drawn from a uniform distribution between 0.2 - 0.6, and 0.6 - 1.0 respectively. This range encompasses the values of K = 0.4 and F = 0.8 recommended in ref. 11.

Performance Metrics. Two important goals in multiobjective evolutionary optimization are to converge to the Pareto-optimal front, and to find multiple, well-distributed

solutions on this front. Here we implement the Y and Δ performance metrics proposed by Deb *et al.* (1,12) to reflect these goals.

The first metric Y measures the extent of convergence to a known set of Pareto-optimal solutions. First, we generate $H = 500$ uniformly spaced solutions from the true Pareto-optimal front in the objective space. For each individual Pareto solution generated with the algorithm, we compute the minimum Euclidean distance to H. The average of these distances represents the metric Y. Obviously, the closer the value of Y is to zero, the closer the convergence is of the algorithm to the true Pareto-optimal front.

While Y directly measures convergence, the metric does not provide information about the spread or diversity of the individual solutions along the front. This is another important performance criterion for evolutionary algorithms for multiobjective optimization problems. We therefore use another metric, denoted as Δ, to measure the extent of spread achieved among the obtained solutions. To calculate this performance metric, we start out by computing the Euclidean distance, d_i between two consecutive nondominated solutions. We use these values to compute the average, $\overline{d}$ of these distances. Thereafter, we fit a curve through the extreme ends of the true Pareto-optimal objective front, and use this curve to extrapolate the generated nondominated solutions with the algorithm to the extreme ends. After doing this, we compute the Euclidean distance d_f and d_l between the extrapolated extreme solutions and the boundary solutions of the obtained nondominated set. We can now compute Δ according to:

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{L} \left| d_i - \overline{d} \right|}{d_f + d_l + (L-1)\overline{d}} \quad (5)$$

where L denotes the number of nondominated solutions with the algorithm. A good distribution would make all distances d_i identical to $\overline{d}$ and would make d_f and $d_l = 0$

resulting in a value of zero for the Δ metric. For more information about these performance metrics please refer to Deb *et al.* (1,12).

Relative Hypervolume Indicator. To quantify how many function evaluations are needed for the individual algorithms to obtain nondominated solutions that are sufficiently close to the true Pareto set, we use the relative hypervolume diagnostic (12,13). This metric evaluates convergence and diversity within a single measure, and is therefore one of the best unary measures available to diagnose whether the non-dominated solution set derived with the algorithm is approximating the true Pareto set. The metric computes the ratio between the objective space dominated by the obtained non-dominated front, and the objective space dominated by the true Pareto set.

Mathematically, for each nondominated solution $i \in P_t$ derived with an algorithm, a hypercube z_i is constructed with reference point W and the solution i as the diagonal corners of the hypercube. Thereafter, a union of all hypercubes is found and its hypervolume (HV) calculated by (13):

$$HV = \text{volume}(U_{i=1}^{|P_t|} z_i) \quad (6)$$

This procedure is repeated for the true Pareto solutions (P^*), and the relative hypervolume (RHV) is then derived from:

$$RHV = 1 - \frac{HV(P_t)}{HV(P^*)} \quad (7)$$

The closer the value of this metric is to zero, the closer the obtained non-dominated set is to the true Pareto-optimal set.

1. Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002) *IEEE Trans. Evol. Comp.* **6,** 182-197.

2. Kennedy, J., Eberhart, R.C. & Shi, Y. (2001) *Swarm Intelligence* (Morgan Kaufmann, CA).

3. Haario, H., Saksman, E. &. Tamminen, J. (2001) *Bernoulli* **7**, 223 – 242.

4. Storn, R. & Price, K. (1997) *J. Global Optimization*, **11(4),** 341-359.

5. Deb, K. & Agrawal, R.B. (1995) *Complex Syst.* **9,** 115-148.

6. Kennedy, J. & Eberhart, R. (1995) *Proc. Fourth IEEE Inter. Conf. on Neural Networks* (IEEE Service Center, Piscataway, NJ).

7. Hu, X., Eberhart, R. & Shi, Y. (2003) *Proc. IEEE Swarm Intelligence Symposium*, 193.

8. Parsopoulos, K.E. & Vrahatis, M.N. (2002) *Proc. of the* 2002 *ACM Symp. Appl. Comp.* (ACM Press, Madrid, Spain).

9. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, H. & Teller, E.J. (1953) *J. Chem. Phys.* **21(6),** 1087-1091.

10. Gelman, A., Carlin, J.B., Stren, H.S. & Rubin, D.B. (1995) *Bayesian data analysis* (Chapmann and Hall, New York, NJ).

11. Iorio, A. & Li, X. (2004) Report downloaded from: http://goanna.cs.rmit.edu.au/~iantony/papers/aus.pdf.

12. Deb, K. (2001) *Multi-Objective Optimization Using Evolutionary Algorithms* (Wiley, New York).

13. Veldhuizen, D.V. (1999) Ph.D. Dissertation (Univ of Ohio, Dayton, OH).