On Multiobjective Optimization

Dr. Ivan Voutchkov,
CEDC, School of Engineering Sciences
Department of Mechanical Engineering
University of Southampton, Highfield, Southampton SO17 1BJ
tel: 023 80 597662
e-mail: iiv@soton.ac.uk

CEDC

OUTP for design

This article may be found at
http://www.soton.ac.uk/~cedc/posters.html

Why multiobjective optimization?

In the world of real engineering design, often there are multiple targets which manufacturers are trying to achieve. For instance in the aerospace industry, a general problem is to minimize weigh, cost and fuel consumption while keeping performance and safety at maximum. Each of those targets might be easy to achieve individually. An airplane made of balsa wood would be very light and will have low fuel consumption, however it will not be structurally strong enough to perform at high speeds or carry useful payload. Also such an airplane would not be safe, i.e., robust to various weather and operational conditions. On the other hand, a solid body and a very powerful engine will make the aircraft structurally sound and able to fly at high speeds, but its cost and fuel consumption will increase enormously. So engineers are continuously solving the problem of making trade-offs and producing designs that will satisfy as many requirements as possible, while industry, commercial and ecological standards are getting ever tighter.

Multiobjective optimization (MO) is a tool to aid engineers choose the best design in a world where many targets need to be satisfied. Unlike conventional optimization, MO will not produce a single solution, but rather a set of solutions, most commonly referred to as Pareto front (PF). By definition it will contain only non-dominated solutions. It is up to the engineer to select a final design by examining this front.

Where is the challenge?

The aim is to produce a well spread out set of optimal designs, with as few function evaluations as possible. There are number of methods published and widely used to do this – MOGA, SPEA, PAES, VEGA, NSGA2, etc. Some are better than others - generally the most preferred in the literature are NSGA2 (Deb) and SPEA2 (Zitzler), because they are found to achieve good results for most problems. The first one is based on genetic algorithms and the second one is based on an evolutionary algorithm, both of which are known to need many function evaluations. In real engineering problems the cost of evaluating a design is probably the biggest obstacle that prevents extensive optimization procedures. In the multiobjective world, the cost is multiplied, because there are multiple expensive results to obtain. Evaluating directly a finite element model can take several days, which makes it impossible to try hundreds or thousands designs.

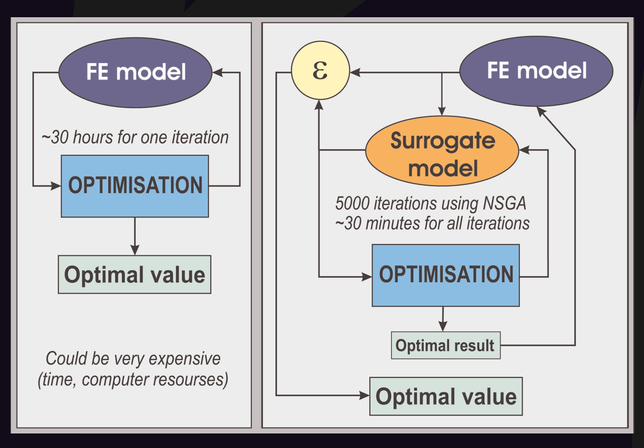

Figure 1 – Using surrogate models for optimization

In the single objective world, approaches using surrogate models are fairly well established and have proven to successfully deal with the problem of computational expense (see Figure 1). Since its introduction, more and more companies have adopted optimization and some are making steps to incorporate this approach in their design cycle as a standard. The reason for this is that instead of using the expensive finite element models, they are substituted with a much cheaper but still accurate replica. This makes optimization not only useful, but usable and affordable.

The key idea that makes surrogate models efficient is that they should become more accurate in the region of interest, rather than equally accurate over the entire design space, as an FE representation will tend to do. This is achieved by adding to the surrogate knowledge base only at points of interest. The procedure is referred to as surrogate update.

Multiobjective optimization using surrogates

Recently, at Southampton University, there has been some progress, using the same idea in the multiobjective world. Using an improved NSGA2 algorithm and the OptionsMatlab interface to the optimization package OPTIONS™, it was possible to create a routine that significantly reduces the number of function evaluations, needed to achieve a high quality Pareto optimal front. The idea has been successfully tested on a number of benchmark test functions, and is currently being applied to real engineering problems. Figure 2 is a striking comparison, demonstrating the idea.

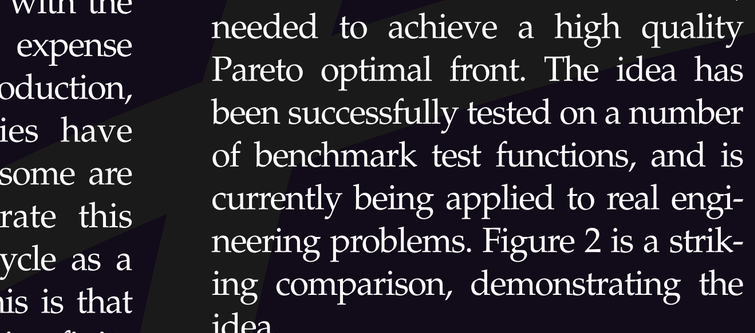

Figure 2
Problem 1: Achieved in 40 evaluations.
Green – Pareto front with surrogates.
Red – solution without surrogates.

The problem has two objective functions and two design variables. The thick green set of diamond symbols represents the Pareto Optimal Solution obtained after 40 full function evaluations using the surrogate technique. This front contains 4500 design points (only 50 are shown) and is identical to the true solution. For the purpose of comparison the red markers represent the solution found with the same number of function evaluations, without using surrogate models. To obtain 900 solutions on the true Pareto front, the latter approach needs 2500 function evaluations. The difference is even more significant if more variables are added – see Table 1.

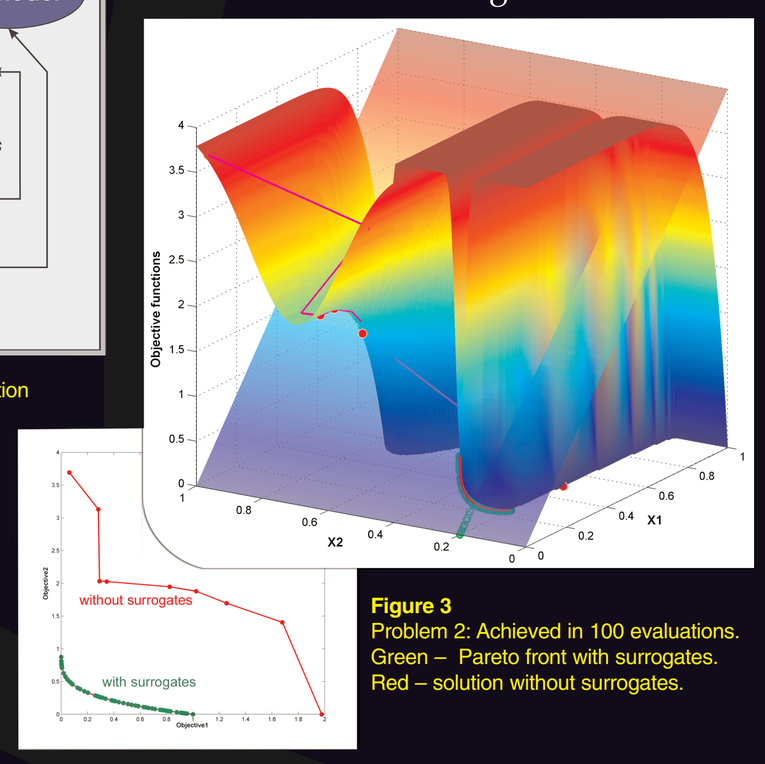

Figure 3
Problem 2: Achieved in 100 evaluations.
Green – Pareto front with surrogates.
Red – solution without surrogates.

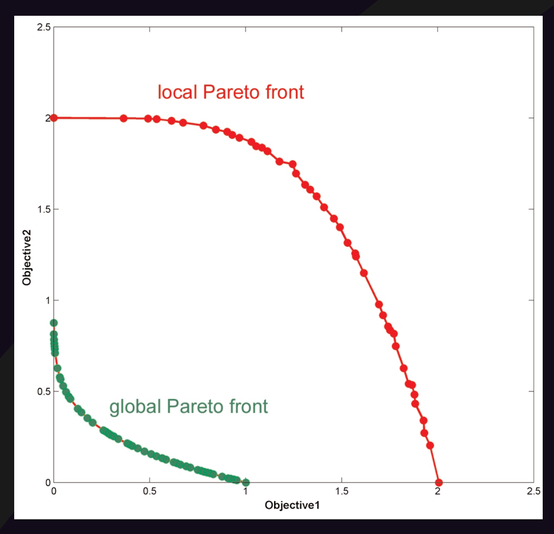

Figure 4 – Escaping from local solutions during the update stage

Number of variables	2	5	10
without surrogates	2500	5000	10000
with surrogates	40	40	60

Table 1 – Full function evaluations for Problem 1 – Figure 2

Problem 2 has two variables and two objective functions. However it has a feature called a 'deceptive Pareto front'. The comparison is similar to the previous example. If surrogate models are not used 2500 function evaluations were needed to get to the same results as with 100 evaluations using surrogates. A technique that will help the algorithm escape from the local Pareto front is used.

As shown in Figure 1, surrogates need updates. In the MO world, a conventional update scheme will choose let's say 20 evenly distributed points on the Pareto front. The selected points will then be calculated using the expensive FE code, the data will be added to the existing set, the surrogates will be retrained and a GA search will be run for a large number of iterations on the surrogates. When the surrogate knowledge base becomes too large, it is filtered, leaving only designs close to the Pareto front. What seems to be a good strategy may fail if the data on which the surrogate is build is filtered down to a local optimum location. One way to prevent this is to include random runs along with those chosen from the pareto front. Another way is to run a second NSGA2 at each update directly on the objective function, ensuring wider spread of update points. Experiments show that the second solution is better than the first one.

It is worth mentioning, that all tools used for this research allow parallel grid computations. The approach described above makes the best of the multiprocessor environment by running all updates simultaneously.

University of Southampton