MULTIOBJECTIVE EVOLUTIONARY ALGORITHMS APPLIED TO AIRCRAFT ENGINE DESIGN

R. Filomeno Coelho*, S. Pierret*, P. Cobas**

***CENAERO A.S.B.L.**
Bâtiment Mermoz 1, 2ème étage – Av. Mermoz, 30 – B-6041 Gosselies (Belgium)
Phone:+32(0)71.91.93.65 / Fax: +32(0)71.91.93.31
E-mail: { rajan.coelho ; stephane.pierret}@cenaero.be

****Empresarios Agrupados Internacional, S.A.***
Magallanes, 3 – 28015 Madrid (Spain)
Phone: +34-91.309.80.28 / Fax: +34-91.591.26.55
E-mail: pce@ecosimpro.com

Keywords: aircraft engine design, optimization, evolutionary algorithms, gas turbine.

Abstract

Within the VIVACE project, an object-oriented simulation tool called PROOSIS is developed to integrate all European gas turbine simulation technology into a single framework. As optimization functionalities are required, the MAX optimization code developed at Cenaero has been made accessible from PROOSIS, and applied for demonstration to a turbofan model.

1 Introduction

In the context of the VIVACE European project dedicated to aeronautical collaborative design [1], the ECP (European Cycle Program) work package develops a comprehensive multi-disciplinary, object-oriented simulation environment in order to integrate all European gas turbine simulation technology into a common framework providing shared standards and methodologies for European universities, research institutes and corporate companies. This simulation tool has been named PROOSIS (i.e.: PRopulsion Object Oriented SImulation Software).

The main goal of PROOSIS is to provide a standard for gas turbine modelling in Europe, and allow non-engine companies to have a common tool for jet engine simulation. As robust and efficient optimization functionalities are also required within its architecture, MAX (an optimization code developed at Cenaero) has been made accessible from PROOSIS. After a description of PROOSIS (§2), MAX optimization algorithms will be described in detail (§3). Then, the coupling of MAX with PROOSIS will be discussed (§4), followed by the conclusions (§5).

2 PROOSIS

As an object-oriented tool, PROOSIS enables the creation of sophisticated engine models by assembling some basic components (characterized by their thermodynamical behaviour). For example, a turbofan (cf. figure 1) can be modelled by assembling compressors, turbines, shafts, a nozzle, a fuel tank, etc. Each individual component is characterized by variables and functions, either internal or external; only the external ones (the ports) communicate with the other components. This philosophy is analogous to object-oriented programming, the internal variables and functions acting as private ones, whereas the ports are accessible by other components like public variables and functions.

Once the engine is designed, steady and transient performance must be calculated for different purposes, like verifying the engine behaviour in the whole flight domain,

calculating fuel consumption for specified missions, etc.

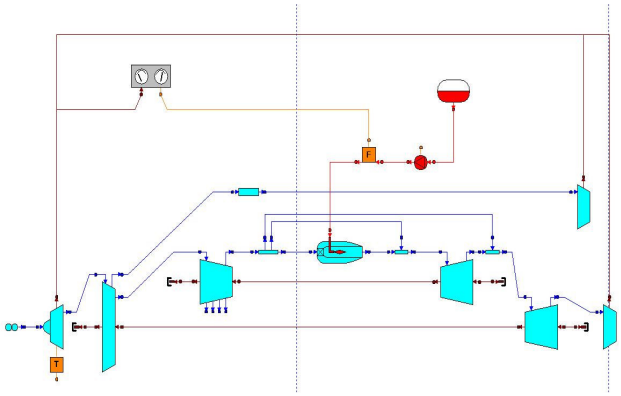

Figure 1: turbofan modelled in PROOSIS.

For instance, Alexiou and Mathioudakis [2] demonstrated the efficiency of PROOSIS in the development and use of an engine performance model for gas turbine.

Then, optimization tools can be used. That matter is discussed in the next section.

3 MAX optimization algorithm

3.1 Multiobjective evolutionary algorithms

A general optimization problem can be formulated as follows:

$$\min_{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}) \quad (1)$$

$$\boldsymbol{g}(\boldsymbol{x}) \geq 0 \quad (2)$$

$$\boldsymbol{h}(\boldsymbol{x}) = 0 \quad (3)$$

$$x_i \in X_i \ \ for\ i = 1,\ldots, n \quad (4)$$

where:

- $\boldsymbol{f}(\boldsymbol{x})^T = [\, f_1(x)\ \ f_2(x) \ldots f_m(x)\,]$; (5)
- $\boldsymbol{g}(\boldsymbol{x})^T = [\, g_1(\boldsymbol{x})\ \ g_2(\boldsymbol{x}) \ldots g_p(\boldsymbol{x})\,]$; (6)
- $\boldsymbol{h}(\boldsymbol{x})^T = [\, h_1(\boldsymbol{x})\ \ h_2(\boldsymbol{x}) \ldots h_q(\boldsymbol{x})\,]$. (7)

Equality constraints are commonly transformed into inequality constraints:

$$h_i(\boldsymbol{x}) = 0 \ \ is\ replaced\ by\ |h_i(\boldsymbol{x})| \leq \varepsilon_i\,, \quad (8)$$

where ε_i is the degree of violation authorized by the user.

Since the late eighties, evolutionary algorithms (EAs) have demonstrated their robustness and efficiency in solving single-objective optimization problems in various engineering fields. Based on the Darwinian law of the survival of the fittest, EAs start with a set of potential designs and make them evolve by successive operations of selection and recombination, aiming at converging towards the global optimum at the end of the process [3].

One of their most common instances is the genetic algorithm (GA) [4]. GAs are particularly well suited for problems with mixed variables; furthermore, they can handle noisy, multi-modal and discontinuous functions. They do not require the computations of the sensitivities, and are more likely to find the global optimum (in comparison with gradient-based algorithms).

However, in lots of industrial applications, dealing with only one objective is generally not sufficient. Therefore, multiobjective evolutionary algorithms (MOEAs) have been developed in order to find the Pareto optimal set, i.e. the set of non-dominated solutions such that there exists no other feasible individual in the search space better with respect to all criteria [5].

The concept of Pareto optimum (or non-dominated solution) constitutes a major key to take the multiobjective aspect into account. By definition, a design vector $\boldsymbol{x}^* \in \boldsymbol{F}$ is Pareto optimal if and only if there exists no other $\boldsymbol{x} \in \boldsymbol{F}$ such that:

$$f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{x}^*) \ \ for\ i = 1,\ldots, m \quad (9)$$

with $f_i(\mathbf{x}) < f_i(\mathbf{x}^*)$ for at least one objective i. $\boldsymbol{F}$ is the feasible domain defined by :

$$\boldsymbol{F} = \{\, \boldsymbol{x} \in \boldsymbol{X} \mid g_j(\boldsymbol{x}) \geq 0 \ \ for\ j = 1,\ldots, p \ \ and\ \ h_k(\boldsymbol{x}) = 0 \ \ for\ k = 1,\ldots, q \,\} \quad (10)$$

As the solution is not unique, the user has to provide additional information about his/her preferences in order to find the optimum solution. Three different approaches are available in the literature [6]:

1. Preferences are used at the end, when the Pareto front has been completely determined (*a posteriori methods*);

2. Preferences are used during the optimization process, in an interactive way (*progressive methods*);
3. Preferences are included since the beginning of the search process (*a priori methods*).

When the user already has a clear opinion about his/her preferences, a priori techniques might give interesting results very quickly [7]. They consist in defining preferences (through weights or via a ranking of the objectives) before the algorithm starts its search. The weighted sum method is the most common instance of these techniques. In this approach, the *m* objective functions are aggregated into one, as follows :

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} w_i \cdot f_i(\boldsymbol{x}) \quad (with \sum_{i=1}^{m} w_i = 1) \quad (11)$$

Other traditional methods include the weighted min-max method, the goal programming, etc. To handle preferences more accurately, specific methods have been developed in the multi-criteria decision aid (MCDA) field, like ELECTRE, PROMETHEE, MELCHIOR, …, and adapted to multiobjective evolutionary optimization [8].

Nevertheless, when the user lacks of information about the problem, a posteriori MOEAs are preferred. Recent advances in MOEAs include NPGA2 (Niched-Pareto Genetic Algorithm 2 [9]), NSGA-II (Non-dominated Sorting Genetic Algorithm II [10]), PAES (Pareto Archived Evolution Strategy [11]) and SPEA2 (Strength Pareto Evolutionary Algorithm 2 [12]).

The SPEA2 method proposed by Zitzler and Thiele [12] is a widespread instance of a posteriori MOEAs. The initial SPEA works as follows:

- *step 1*: generate an initial population P and create the empty external non-dominated set P_0;
- *step 2*: copy non-dominated members of P to P_0;
- *step 3*: remove solutions within P_0 which are covered by any other member of P_0;
- *step 4*: if the cardinality of P_0 exceeds a user-defined number N_0, prune P_0 by means of clustering;
- *step 5*: calculate the fitness of each individual in P and P_0 ;
- *step 6*: select individuals from P + P_0 until the mating pool is filled, and apply crossover and mutation;
- *step 7*: if the maximum number of generations is not reached, go to Step 2.

In order to remediate to some drawbacks of SPEA, Zitzler et al. proposed to change the fitness calculation in the so-called SPEA2 method (the reader interested in these modifications can find the detailed information in [12]). The numerical results presented in [12] show that in all test cases SPEA2 performs better than SPEA.

3.2 Approximation methods

However, one of the main drawbacks of a posteriori MOEAs is the large number of function evaluations they require, which can be critical when time consuming computations have to be performed. Therefore, the method implemented in MAX (the optimization code developed in Cenaero) is organized as follows (cf. figure 2): first, a database is built by running the exact model on a set of initial individuals, and used to create an approximated model by means of radial basis function networks. Then, a predefined number of loops are performed, each loop consisting in:

1. executing SPEA2 on the approximated model;
2. computing a subset of the Pareto solutions found by the EA thanks to the accurate (expensive) model;
3. adding these new points to the database in order to construct a better approximation.

This approach has given very good results while decreasing the number of function calls by a factor close to 10 (in comparison with SPEA2 without approximation), opening the path of multiobjective optimization to many areas of engineering. Indeed, MAX has been

used successfully in several applications (aerodynamic optimization of turbomachinery blades [13], heat pipe design optimization, etc.) involving expensive simulations (CFD, FEM, etc.).

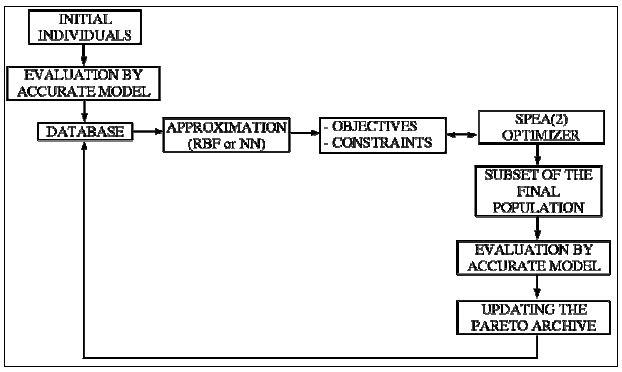

Figure 2: flow-chart of the multiobjective optimization algorithm implemented in MAX.

Of course this approach is also available for single-objective optimization, in a slightly modified way, as depicted in figure 3.

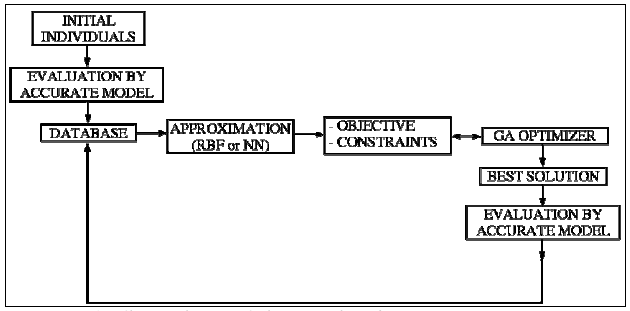

Figure 3: flow-chart of the single-objective optimization algorithm implemented in MAX.

3.3 Handling of the constraints

In addition to these developments, specific constraint-handling techniques have been added in conjunction with these methods in order to enforce the algorithm to reach the feasible search space (i.e. the region where the designs satisfy all the technical requirements imposed by the user).

The handling of constraints in genetic algorithms is a delicate task, which explains the important amount of techniques devised to address it. In single-objective optimization, Michalewicz and Coello classified them in six categories [14]:

1. lethalization or death penalty techniques;
2. penalization techniques;
3. methods separating the objectives and the constraints;
4. methods with decoders or constraint-preserving operators;
5. repair algorithms;
6. hybrid methods.

The penalization of individuals violating the constraints is the most popular approach by the EA community. A new objective function $f'(\boldsymbol{x})$ is defined by adding a penalty to each infeasible solution.

For instance, Joines and Houck proposed a method with dynamic coefficients, whose values are computed with respect to the number of the current generation [15]:

$$f'(\boldsymbol{x}) = f(\boldsymbol{x}) + (C.t)^{\alpha} \sum_{j=1}^{p} G_j \left| g_j(\boldsymbol{x}) \right|^{\beta} \quad (12)$$

where G_j is the Heaviside operator such that $G_j = 1$ when the j^{th} constraint is violated ; t is the generation number, and C, α and β are parameters of the method (standard values recommended in [15] are: C=1, α=1 and β=2).

Penalization techniques provide good results without significant modification of the standard evolutionary algorithm. However, the difficulty in the choice of the parameters constitutes their main drawback, because no general rule can be applied to determine their values.

To alleviate this problem, Deb proposed a constraint tournament selection method that works as follows [16]:

- when two individuals are compared, the feasible one is always preferred to the infeasible one;
- if both individuals are feasible, the one with the best objective function value is chosen;
- if both individuals are infeasible, the one with the least violation of the constraints is preferred.

This technique has demonstrated to be successful [16], especially in providing feasible solutions in highly constrained problems.

Both methods (Joines and Houck's and Deb's) have been implemented in MAX GAs. However, the handling of the constraints must not only be performed inside the GA, but also during the calculation of the global fitness function used for validation (which is required to compare the best solutions found by the GA).

In this case, it is thus necessary to construct a global fitness function taking into account the constraints, as it is the case in penalization. Therefore, three different penalty methods have been implemented:

- static penalty: the fitness function is computed as follows:

$$f'(\boldsymbol{x}) = f(\boldsymbol{x}) + C_j \sum_{j=1}^{p} G_j \left| g_j(\boldsymbol{x}) \right| \qquad (13)$$

- Joines and Houck's dynamic penalty (see above);
- Bean and Hadj-Alouane's adaptive penalty [16]:

$$f'(\boldsymbol{x}) = f(\boldsymbol{x}) + \lambda(t) \sum_{j=1}^{p} G_j \left| g_j(\boldsymbol{x}) \right|^2 \qquad (14)$$

where:

- $\lambda(t+1) = \lambda(t)/\beta_1$ if case #1;
- $\lambda(t+1) = \beta_2.\lambda(t)$ if case #2;
- $\lambda(t+1) = \lambda(t)$ otherwise,

with cases #1 and #2 denoting situations where the best individual in the last k generations was always (#1) or never (#2) feasible (k must be defined by the user). It means that the penalty decreases if the best individual in the last k generations was systematically feasible, and decreases if it was always infeasible; otherwise the penalty parameter remains unchanged.

Adaptive penalization has the advantage to pay heed to the results obtained at previous generations.

3.4 Numerical examples

In order to demonstrate MAX ability to solve optimization problems, two examples will be presented.

First, Rastrigin 2D single-objective function is tested. It has several local minima, as depicted in fig. 4 and 5 (zoom), and its global optimum is characterized by an objective function $f(\boldsymbol{x}^*)$ equal to 0.

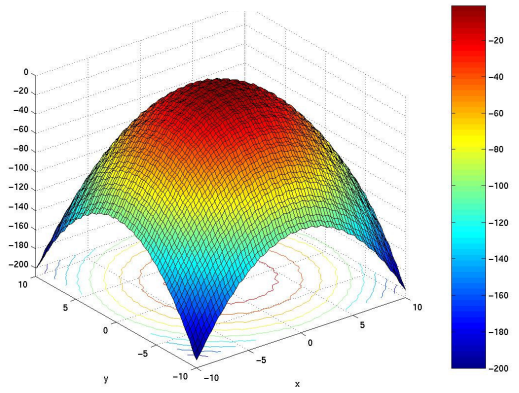

Figure 4: Rastrigin 2D function.

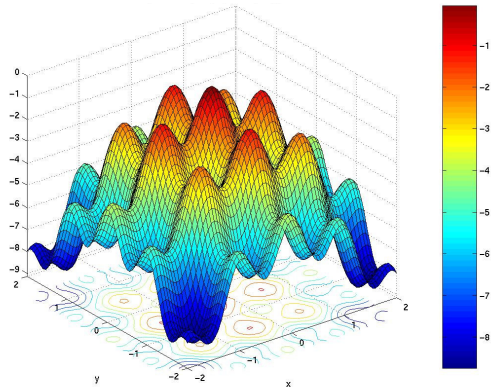

Figure 5: Rastrigin 2D function (zoom).

MAX *Approximated GA* has been applied with different numbers of design loops (1 design loop means: building the approximation model + optimizing this model + computing the solution with the accurate model). The results are collected in table 1, showing that acceptable solutions can be found for a smaller number of function evaluations than with a classical genetic algorithm (without approximation).

Test case	*Rastrigin 2D*
Classical GA	
Number of evaluations	1562
Best objective function	0.0
Approximated GA-25	
Number of evaluations	30
Best objective function	1.8304
Approximated GA-50	
Number of evaluations	55
Best objective function	0.1882
Approximated GA-75	
Number of evaluations	80
Best objective function	0.0264
Approximated GA-100	
Number of evaluations	105
Best objective function	$2.364.10^{-6}$
Approximated GA-200	
Number of evaluations	205
Best objective function	$3.0816.10^{-11}$

Table 1: MAX results for Rastrigin 2D function.

The second example, due to Srinivas [17], is characterized by 2 objectives and 2 constraints.

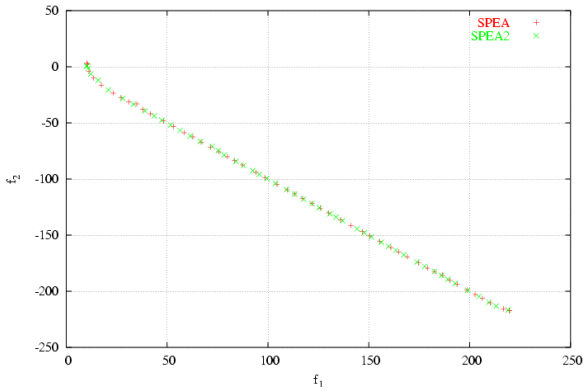

Figure 6: MAX SPEA and SPEA2 results for Srinivas problem.

Figures 6 and 7 depict the Pareto fronts obtained without or with approximation scheme, which are similar. This shows clearly the benefit of using MAX *Approximated-GA* to decrease the number of evaluations required to find the Pareto front, since the classical SPEA(2) method needed 2500 evaluations to locate the Pareto front, while the *Approximated-GA* found it in only 493 evaluations.

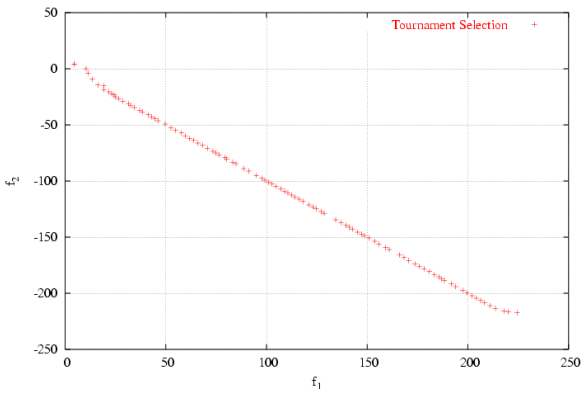

Figure 7: MAX Approximated-GA results for Srinivas problem.

3.5 MAX API for PROOSIS

In addition to its advanced functionalities for engine performance modelling, PROOSIS software is meant to act interatively with other tools and softwares. In particular, C++ and Fortran codes can be coupled with PROOSIS as soon as an application programming interface (API) is provided.

Consequently, as MAX is built on an C++ object-oriented architecture, it has been converted as an API class to allow for fast and efficient integration within PROOSIS. Practically, it means that within PROOSIS, all classes of MAX defined in the API are accessible.

The next section describes the step-by-step procedure followed to perform optimization tasks in a model defined in PROOSIS.

4 Applications

In order to understand the flow of events that must be performed to optimize a PROOSIS model, an example taken from the GAS TURBINE EXAMPLES library will be used: the turbofan. Its graphical representation in EcoDiagram has been illustrated in figure 1.

The efficiencies of the fan and of the compressor are the variables, and the goal is to maximize the thrust. No constraints are imposed. This trivial example is meant to demonstrate the ability of MAX API C++ class

to be used in conjunction with PROOSIS models.

The first step consists in building experiment function, which will contain the definition of the objectives and constraints to optimize.

```
FUNCTION NO TYPE fcn turbofan
   (OUT INTEGER nDesignVariables,
    OUT REAL designVariables[],
    OUT REAL responseVariables[],
    OUT INTEGER successSwitch)
BODY
  Compressor.EP=designVariables[1]
   Fan.EP = designVariables[2]
  TIME = 0.
  STEADY()
  responseVariables[1]=
     Nozzle.thrust.F
  successSwitch= 1
END FUNCTION
```

The *designVariables* are real (continuous) variables, whereas the *responseVariables* are the functions (objectives and constraints) needed by the optimization problem. After the definition of a function computing the equations governing the behaviour of the component and defining the objectives and constraints, the variables and parameters of the optimization have to be defined, as well as an object of *maxNewAPI* class (which will be called to optimize the turbofan).

```
EXPERIMENT optimization ON
   turbofan.design
DECLS
-- setting the parameters for MAX
   optimization
INTEGER nDesignVariables = 2
INTEGER nResponseVariables = 1
INTEGER nObjectives = 1
INTEGER objectiveType[1]= {1}
  -- maximize
INTEGER nConstraints = 0
  -- no constraints
INTEGER constraintType[2]= {1,1}
  -- upper bound
INTEGER optimizationAlgorithm = 1
INTEGER reproductionCycleCount= 20
INTEGER populationSize = 20
INTEGER designLoopCount= 10
REAL designVariables[2]= {0.8,0.8}
REAL designVariablesLower[2] =
     {0.75,0.75}
REAL designVariablesUpper[2] =
     {0.95,0.95}
REAL constraintBound[2] = {0,0}
REAL solutionSet[1000]
OBJECTS
   maxNewAPI opti
   -- defining an object opti of
      class maxNewAPI

INIT
  -- set initial values for
     variables
  (...)
BOUNDS
  -- set expressions for boundary
     variables: v = f(t,...)
  (...)
BODY
  -- launching the optimization
  opti.max(fcn turbofan,
     nDesignVariables,
     designVariables,
     designVariablesLower,
     designVariablesUpper,
     nResponseVariables,
     nObjectives,
     objectiveType,
     nConstraints,
     constraintType,
     constraintBound,
     solutionSet,
     optimizationAlgorithm,
     reproductionCycleCount,
     populationSize,
     designLoopCount)
END EXPERIMENT
```

In the declaration field, all the variables required by the optimization must be set.

Here are some remarks about three variables:

- *solutionSet*: this vector contains the solution found by the algorithm. In single-objective optimization, it consists of a vector whose first elements are the design variables, followed by the objectives (and possibly the constraints). In multiobjective optimization, the format is the same as for 1 objective, except that in general the solution is made of multiple (non-dominated) points constituting an approximation of the Pareto set;
- *optimizationAlgorithm*: the user can choose among 4 optimization algorithms:
  - 1: single-objective genetic algorithm (with advanced operators);
  - 2: single-objective genetic algorithm combined with an approximated model based on

radial basis function networks (to speed up the process when time consuming models are used);
    - 3: multiobjective genetic algorithm based on SPEA;
    - 4: multiobjective genetic algorithm based on SPEA and combined with an approximated model.
- *designLoopCount*: when the GA is combined with an approximated model, the process is divided in two levels: a general level consisting in building the approximated model and running the exact simulation, and a local level (the GA applied to the approximated model). The number of design loops corresponds to the number of runs of the GA on the approximate models.

Once the parameters of the problem are defined, and an object *opti* created, the optimization process can be launched via the GUI in PROOSIS.

Predictably, the results show that the maximum thrust is obtained for compressor and fan with maximum efficiencies.

```
MAX optimization results
------------------------
Best design vector: component 1
   = 0.950000
Best design vector: component 2
   = 0.950000
Corresponding value of objective 1
   = 57340.344827
```

Should the solution found by the algorithm be infeasible, this would be indicated in the report. In all cases, when there are constraints, their values for the best design are written.

In multiobjective optimization, the same formats are used, except that the solution is generally not composed of 1 point but of several non-dominated points. Regarding the constraints, the number of feasible and infeasible solutions in the Pareto front are written in the report.

More sophisticated and realistic applications will be treated during the last period of the VIVACE project; this first example has been presented in order to illustrate the coupling of MAX and PROOSIS and its promising application to the optimization of gas turbine engine performance models.

5 Conclusions

This work focuses on the coupling of MAX optimization software to engine performance models created thanks to PROOSIS, an object-oriented simulation software dedicated to gas turbine applications. The optimization proposed in this study combines single- and multiobjective evolutionary algorithms with approximation methods in order to reduce the number of calls to the computation of the model. This approach gives similar results (in comparison with multiobjective schemes without approximation) while decreasing drastically the number of function evaluations.

In this paper, the emphasis has been put to illustrate the coupling of MAX with PROOSIS, allowing for performing optimization tasks in a straightforward way. As PROOSIS is still in development within the VIVACE project, more realistic examples will be treated eventually.

Acknowledgements

CENAERO A.S.B.L. is supported by the Walloon Region and the ERDF European funds (under contract EP1A 122030000102).

This work in particular has been funded through the Integrated Project VIVACE (AIP3-CT-2003-502917).

The authors would also like to warmly thank the partners of Work Package 2.4 of VIVACE for their collaboration and support.

References

[1] http://www.vivaceproject.com

[2] A. Alexiou, K. Mathioudakis, *Gas Turbine Engine Performance Model Applications Using an Object-oriented Simulation Tool,* Proceedings of ASME TurboExpo 2006, May 8-11, Barcelona, Spain (2006).

[3] T. Bäck T, D.B. Fogel, Z. Michalewicz (eds.), *Handbook of Evolutionary Computation*, Oxford University Press, New York, 1997.

[4] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman, New York, 1989.

[5] D.A. Van Veldhuizen, G.B. Lamont, *Multiobjective*

Evolutionary Algorithms : Analyzing the State-of-the-Art, Evolutionary Computation vol. 8 (2), pp. 125-147 (2000).

[6] C.A.C. Coello, D.A. Van Velduizen, G.B. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*, Kluwer Academic/Plenum Publishers, New York, 576 pp. (2002).

[7] R. Filomeno Coelho, *Multi-criteria Optimization with Expert Rules for Mechanical Design*, Faculty of Applied Sciences, Université Libre de Bruxelles (2004).

[8] R. Filomeno Coelho, H. Bersini, Ph. Bouillard, *Parametrical Mechanical Design with Constraints and Preferences: Application to a Purge Valve*, Computer Methods in Applied Mechanics and Engineering, 192/39-40, pp. 4355-4378 (2003).

[9] M. Erickson, A. Mayer and J. Horn, *The Niched Pareto Genetic Algorithm 2 Applied to the Design of Groundwater Remediation Systems*, In E. Zitlzler, K. Deb, L. Thiele, C.A.C. Coello and D. Corne, editors, First International Conference on Evolutionary Multicriterion Optimization, pp. 681-695, Springer-Verlag, Lecture Notes on Computer Science No. 1993 (2001).

[10] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, *A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II*, IEEE Transactions on Evolutionary Computation, vol. 6 (2), pp. 182-197 (2002).

[11] J.D. Knowles and D.W. Corne, *Approximating the Non-dominated Front Using the Pareto Archived Evolution Strategy*, Evolutionary Computation, vol. 8 (2), pp. 149-172 (2000).

[12] E. Zitzler, M. Laumanns, L. Thiele, *SPEA2: improving the strength Pareto evolutionary algorithm*, EUROGEN 2001, K. Giannakoglou (ed.), CIMNE Editions (2001).

[13] S. Pierret, H. Kato, R. Filomeno Coelho, A. Merchant, *Multi-objective and Multi-disciplinary shape optimization*, EUROGEN 2005, Evolutionary and Deterministic Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems, R. Schilling, W. Haase, J. Périaux, H. Baier, G. Bugeda (eds.) (2005).

[14] C.A.C. Coello, *Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art*, Computer Methods in Applied Mechanics and Engineering 2002; 191:1245-1287.

[15] J.A. Joines, C.R. Houck, *On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs*, Proceedings of the First IEEE International Conference on Evolutionary Computation, pp. 579-584, IEEE Press (1994).

[16] K. Deb, An efficient constraint handling method for evolutionary algorithms, Computer Methods in Applied Mechanics and Engineering, 186, pp. 311-338 (2000).

[17] K. Deb, Constrained Test Problems for Multi-Objective Evolutionary Optimization, First International Conference on Evolutionary Multi-Criterion Optimization, pp. 284-298 (2001).