Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 108C (2017) 1443–1452

Procedia Computer Science

International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland

Multi-objective optimisation in scientific workflow.

Hoang Anh Nguyen[1*], Zane van Iperen[1], Sreekanth Raghunath[2,4], David Abramson[1], Timoleon Kipouros[3], Sandeep Somasekharan[4]

[1]*Research Computing Centre, The University of Queensland, Australia.*
[2]*Centre for Hypersonics, The University of Queensland, Australia*
[3]*Propulsion Engineering Centre, Cranfield University, U.K.*
[4]*Zeus Numerix Pvt. Ltd. Pune, India.*

Abstract

Engineering design is typically a complex process that involves finding a set of designs satisfying various performance criteria. As a result, optimisation algorithms dealing with only single-objective are not sufficient to deal with many real-life problems. Meanwhile, scientific workflows have been shown to be an effective technology for automating and encapsulating scientific processes. While optimisation algorithms have been integrated into workflow tools, they are generally single-objective. This paper first presents our latest development to incorporate multi-objective optimisation algorithms into scientific workflows. We demonstrate the efficacy of these capabilities with the formulation of a three-objective aerodynamics optimisation problem. We target to improve the aerodynamic characteristics of a typical 2D airfoil profile considering also the laminar-turbulent transition location for more accurate estimation of the total drag. We deploy two different heuristic optimisation algorithms and compare the preliminary results.

Peer-review under responsibility of the scientific committee of the International Conference on Computational Science

Keywords: Multi-objective optimisation; Scientific Workflow; Engineering Design.

1 Introduction

Engineering design typically involves searching for good solutions that meet various performance criteria and constraints. These optimisation problems are complex because of several characteristics. First, they typically involve more than one, and often conflicting, objective functions. Although objective functions can be aggregated into a single one and thus simplifying the problem, multi-objective optimisation is generally considered to be more effective than single objective because it allows more flexible trading between objectives [1]. Second, the evaluations of objective functions might be computationally intensive and time consuming, which often requires access to supercomputers. Additionally, the optimisation domain can be large, making the whole optimisation

* Corresponding author: uqhngu36@uq.edu.au

1877-0509
Peer-review under responsibility of the scientific committee of the International Conference on Computational Science
10.1016/j.procs.2017.05.213

process even more computationally intensive and time consuming. The third characteristic is related to the implementation details of the optimisation algorithms. As global optimisation is known to be NP-complete, heuristics are often required to find good solutions in a reasonable time [2]. The last several decades have seen much investment in meta-heuristics, especially nature-inspired ones such as simulated annealing and genetic algorithms [3]. This results in many different implementations of continuously appeared meta-heuristics, which creates another level of *complexity* for users when dealing with optimisation problems in terms of problem formulation, benchmarking, etc.

Scientific workflow technology has been shown to be effective for automating and encapsulating scientific processes [4]. Scientific workflow engines simplify the programming task by providing a high-level environment in which users connect a set of previously defined components, implementing a computational pipeline that meets their needs. These engines are typically integrated with distributed computing middleware, allowing workflows to distribute computation intensive jobs to high-performance computing platforms. There have been a large number of workflow engines produced in recent years, but all provide similar capabilities and functions. For a fairly recent view, we refer the reader to [3].

While optimisation codes are traditionally monolithic, solving optimisation problems with scientific workflows brings several benefits. First, the workflow can expose the components of the optimisation process to the user [5], making it is relatively straightforward to replace existing or add new components. Second, the optimisation can use distributed computing support embedded in scientific workflow engines, allowing computational intensive jobs to be off-loaded to high performance machines. With these benefits, scientific workflows have been used to formulate and solve optimisation problems [5]–[7]. However, only single objective optimisation is currently supported by these systems.

In this paper, we present our latest developments that enable multi-objective optimisation. Importantly, we aim to develop an *extensible* framework for future integration of different optimisation algorithms. We demonstrate the extensibility of our design by integrating two implementations of popular multi-objective optimisation algorithms. The paper then presents the solution of a state-of-the-art airfoil shape optimisation that employs a new methodology resulting in improved estimation of the total drag. Details of the methodology are presented later in Section 4.

2 Background

Optimisation is the process of searching a parameter space for the good solutions to a problem that is defined by one or multiple objective functions. In case of only one objective function, the solution is a point in the search space such that the objective function achieves its optimal (minimal or maximal) value [8]. When there is more than one objective function, it is unlikely to have a single point that is optimal for all the functions, especially if there are conflicting objectives. Instead, the solution for these problems is a set of Pareto optimal points [9]. Each Pareto optimal point is non-dominant to other point because the value of one object cannot be improved further without scarifying other objects.

As mentioned earlier, multiple objectives can be aggregated into single objective, often by a weighted sum of the objectives [8]. This allows single-objective optimisation techniques to be used to optimise the composite objective function. However, it requires the composite function to be pre-defined and the optimisation process generates a single vector as the optimal solution. This approach is not ideal in real-life use cases because the composite function reflects the designers' assumptions about the trade-off between objective functions, rather than the actual trade-offs [1]. As a result, multiple-objective optimisation is better to solve real-life optimisation problems.

One class of optimisation algorithm that attracts research interests is meta-heuristics. As opposed to problem-specific heuristics, meta-heuristics are algorithms designed to solve a range of problems

without requiring significant adaptation to each problem [10]. Most meta-heuristic algorithms share a number of characteristics. First, they are typically stochastic, rather than deterministic as in classic optimisation algorithms [11], [12]. Second, they do not make use of gradient information of objective functions to guide the search. The gradient-based methods are found ill suited for real world multi-objective optimisation problems due to large search space with many local minima [1]. Instead, the search is guided based on some nature-inspired principles from physics, biology, etc. Meta-heuristics algorithms can be classified into two main groups: *trajectory-based* (or *single-solution*) and *population-based*. Trajectory-based approaches only deal with single candidate solution. They start with only one candidate solution, and then improve on that solution, describing a trajectory in design space [12]. Examples of trajectory-based approach are simulated annealing, tabu search, etc. On the other hand, population-based algorithms often use population characteristics to guide the search [12]. Some popular algorithms in this group are evolutionary algorithms, genetic algorithms and particle swarms.

The great interest in optimisation in general has created a large number of implementations of these algorithms. These implementations are often very different in parameter space specification, the programming languages, the supported algorithms, etc. Various frameworks have been created to standardize some of these aspects, thus easing the process of solving optimisation problems. Some examples of those frameworks are: jMetal [11], OPT4J [13] and HeuristicLab [14]. These frameworks share two common characteristics. *First*, they separate the optimisation algorithms from the optimisation problem, allowing different optimisation implementations to be used on the same problem. Users generally need to write problems as plug-ins to the frameworks. S*econd*, these frameworks are generally extensible, allowing new meta-heuristic algorithms to be integrated.

Workflow technology has also been used to solve optimisation problems. Compared to monolithic codes, workflows provide users with a clear view of data flow within the optimisation process, and thus make it easier to substitute those components [5]. There exist some work in the scientific workflow community that recognizes the advantage of formulating and solving optimisation problems using workflows. Crick *et al.* [7] added an *optimizer* component into Taverna workflow engine to support its multi-disciplinary optimisation use case. This *optimizer* is, however, specific to structural optimisation. Geodise [6] is capable of representing optimisation problems in workflows. Nimrod/OK augments Kepler with a similar functionality [5]. Both systems are similar in how they implement optimisation workflows. First, they both use loops to represent optimisation process, which is directed by an optimisation component being the optimisation algorithm. Second, both packages support the distribution of heavy computation jobs to various HPC platforms. Both systems, however, only supported single-objective optimisation.

3 Multi-objective optimisation in scientific workflow

As discussed, the main objective of this paper is to develop an extensible framework to support multi-objective optimisation workflows. In order to reduce the development time, we decided to base our design on Nimrod/OK, an existing framework that supports single objective optimisation. This section first describes how optimisation workflows are implemented in Nimrod/OK, and then explains how our development enables multi-objective optimisation workflows. We also decided to integrate two multi-objective optimisation implementations: tabu search (a trajectory-based meta-heuristic algorithm) and GA (a population-based meta-heuristic algorithm). Notably, these two implementations are in different programming languages, which partly show the extensibility of our design.

3.1 Nimrod/OK

Nimrod/OK is an optimisation suite built on top of Kepler, an open source workflow engine written in Java. Kepler inherits the actor-oriented modeling approach from a mature, dataflow-oriented

platform called Ptolemy II [4]. While there is various middleware has been integrated to Kepler, Nimrod/OK uses Nimrod/G to offload computational jobs to high performance computing platforms [5].

A typical optimisation workflow in Nimrod/OK is shown in Figure 1. First, the search domain is defined by the *DomainSpecification* component. Within this domain, *PointsGenerator* selects starting points for the optimisation. Each starting point then starts a new search depending on the algorithm implemented by *Optimiser*. The search algorithm will generate a set of points to be evaluated at *Objective Evaluator* if they pass the *Constraint Enforcer* component. Both *Constraint Enforcer* and *Objective Evaluator* are user-defined components. The results from the evaluations are then passed back to the search algorithm for next iterations. The optimisation process only stops when a certain condition is reached, for instance reaching maximum number of iterations, or some convergent criteria

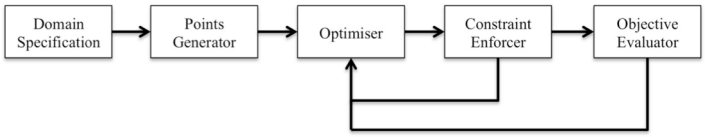


Figure 1: Nimrod/OK optimisation process [5]

have been met.

Each optimisation algorithm in Nimrod/OK is implemented as a separate *Optimiser* actor. At the time the paper is written, Nimrod/OK supports four algorithms: Simplex, Subdivision, Hooke and Jeeves [5] and a in-house development of single objective genetic algorithm [15].

3.2 Optimisation Actor

The current development aims to achieve three objectives. *First*, it needs to support multi-objective algorithms. Regarding the process shown in Figure 1, the Optimiser needs to receive results from more than one objective function. These results are then need to be passed to the optimisation algorithm in order to determine the next evaluation points. *Second,* we aim to provide all the implementations within a single generic actor. Switching between different algorithms is done within this *Optimisation*

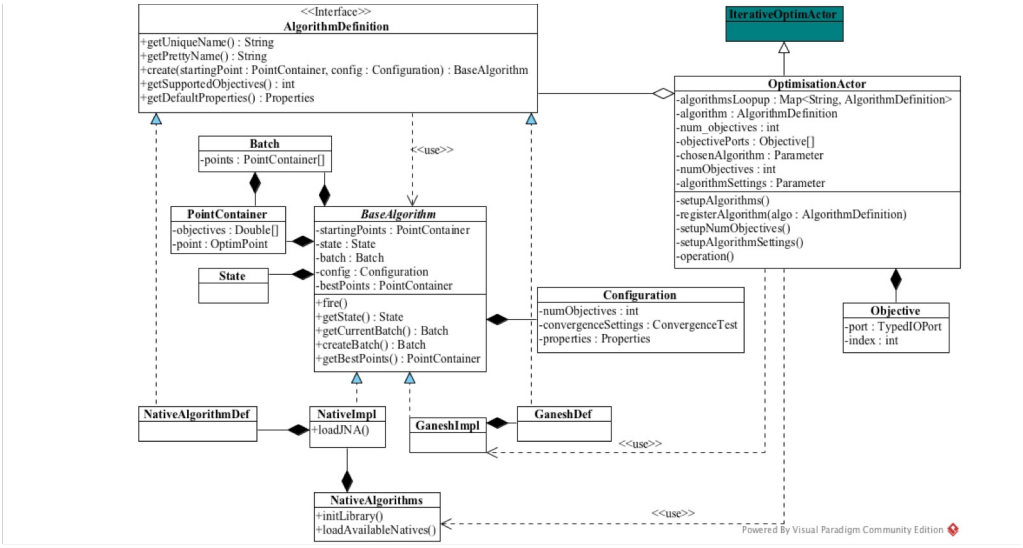


Figure 2: Java interface of the Optimisation Actor.

Actor, resulting in simpler switch between algorithms. *Third,* we aim to support cross-language implementations of optimisation algorithms. The initial target is to support Java and C/C++ implementations due to their popularity.

Figure 2 shows the class diagram of the framework. The new *Optimisation Actor* inherits from Nimrod/OK's existing *IterativeOptimActor*. Each implementation of optimisation algorithm needs to implement the *AlgorithmDefinition* interface, which defines its name, the number of supported objectives and other default properties. The *Optimisation Actor* then *register*s these algorithms when it is initialised. This actor allows users to perform two operations: 1) switch between registered algorithms and 2) specify the number of objective functions. The latter operation generates a number of input ports corresponding to the number of specified objective functions. These ports receive the results of objective function evaluations and pass it to the corresponding algorithm. During the execution, every iteration of the optimisation workflow corresponds to one iteration of the search process.

Java Native Access [16] is used to support multiple programming languages. JNA is the glue between the Java interface and the underlying native implementations. Through JNA, the *Optimisation Actor* queries the available algorithms and instantiates a Java "wrapper" class for each, which is then registered alongside the other Java-based implementations. At run-time, JNA will handle the serialisation and deserialisation of data structures from Java to C and vice versa, facilitating transparent usage of native algorithm implementations.

3.3 Integration of multi-objective optimisation algorithms

We integrate two implementations of popular multi-objective optimisation algorithms into the new Optimisation Actor. One population-based algorithm and one trajectory-based algorithm were selected to show support for both types of meta-heuristics. We also choose one C/C++ implementation to show the cross-language support. These two multi-objective implementations are both available from the Optimisation Actor.

Ganesh [17] is a Java-based framework that offers a multi-objective GA-based optimisation algorithm. The framework consists of two main packages: core GA and plugins. While the core package implements the optimisation algorithm itself, the plugins package provides interfaces and classes to specify optimisation problems. Optimisation problems are specified as a plugins with the domain of the optimisation and evaluations of objective functions. The evaluations can be done within the same process or spawned as sub-processes.

The optimisation algorithm implemented in Ganesh is a GA-based meta-heuristic with self-adaptation of its control parameters [17]. Control parameters are encoded in the internal representation of each candidate solution along with the main parameters, which are problem definition parameters applying to the objective functions. These control parameters are subject to change along with the main parameters due to mutation and crossover. This kind of adaptation is different from those that are instigated algorithmically by feedbacks at the higher level of the GA.

As Ganesh is written in Java, the integration is quite straightforward. A wrapper class is created that implements the *BaseAlgorithm* class. This *GaneshImpl* class makes use of Ganesh core part to perform the optimisation process. The implementation's name and attributes are specified in the *GaneshDef* class. This is partially shown in Figure 2.

MOTS2 is based on the Tabu search algorithm proposed by Connor and Tilley [8]. This algorithm couples a local search algorithm with short, medium and long-term memories to implement search intensification and diversification [8]. At any given point, the algorithm selects a new point in the search space to be the next current point based on the Hooke and Jeeves move. The short-term memory is used to store the recently visited points; the algorithm is not allowed to revisit these points (they are 'Tabu'). The medium-term memory records optimal or new-optimal points, which are used to focus the search on known good values of objective functions. Finally, the long-term memory is used to store information about explored regions; the information is then use to diversify the search to

under-explored regions. Compared to the original Tabu search, MOTS2 modifies several key areas, including H&J move, search intensification and restart strategy. The details of MOTS2 implementation is described in [8].

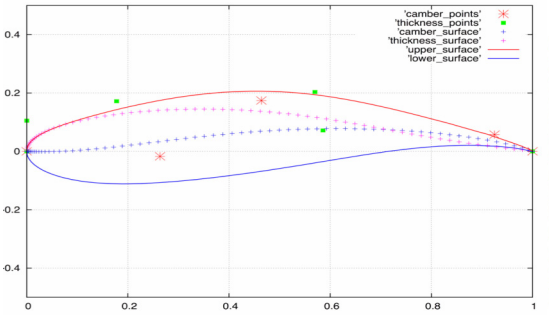
MOTS2 was originally implemented as a stand-alone application/library. It is then later integrated into the Nimrod/O optimisation framework [18]. For this implementation, MOTS2 is bundled in a native library that is loaded by the *Optimisation Actor*. Once loaded, the optimisation functionality is then exposed to the actor via JNA. Each iteration of the optimisation workflow corresponds to one iteration in the search process.

4 Airfoil Shape Optimisation

So far, we have explained our framework and the two existing implementations of multi-objective algorithms integrated into the framework. In this section, a state-of-the-art airfoil optimisation is used to demonstrate how the newly developed components are used to develop an aerodynamic shape optimisation of two-dimensional airfoils.

Viscous drag reduction in aerospace vehicles has always been one of the most challenging problems in aerospace research. Various techniques employed for viscous drag reduction have been described in [19], [20]. One of the popular approaches adopted by researchers to reduce drag is to maximise the extent of laminar flow over the wings. This is achieved either by shape optimisation [21]–[23] or using one of laminar flow control techniques, for instance, adding riblets on the surface of the wing [24], wall heating [25] or suction [26]. The novelty in this airfoil shape optimisation is the inclusion of skin friction through the transition zone for the total drag estimation. Most of the other reported optimisation studies either assume that the flow would be fully turbulent downstream of the location of onset of transition or do not include the effects of transition in their studies at all [27], which results in sub-optimal designs.

Airfoil geometries in our optimisation process are defined by two Bezier curves: one representing thickness curve and the other one representing camber curve of the airfoil. Each curve is in turned defined by Bezier control points. We use six thickness points to specify thickness curve and five camber points to specify camber curve (Figure 3 left). As the first and last points in both thickness and camber curve are fixed, the movement of the three camber points and four thickness points in the Cartesian space define the parameter space of the optimisation process (Figure 3 left). The upper and lower surface of the airfoil is then generated based on the camber and thickness curve. A check is performed to see if the generated airfoils lie within the geometry constraints specified and only valid individuals are carried on to the next step. The thickness range of the generated airfoils is limited to remain between 7% and 50% of the chord at any given location. To ensure a smooth leading edge, the thickness is limited to 11% of the chord in the first 5% of the chord near the leading edge. The maximum thickness is limited to 20% of the chord in the last 20% of the chord and to 6% of the chord in the last 6% of the chord. The thickness at all locations is limited to above 2%, ensuring non-zero



parameter ax2 float range from 0 to 0.4;
parameter ax3 float range from 0.2 to 0.6;
parameter ax4 float range from 0.5 to 1.0;
parameter ay1 float range from -0.1 to 0.3;
parameter ay2 float range from -0.1 to 0.3;
parameter ay3 float range from -0.1 to 0.3;
parameter ay4 float range from -0.1 to 0.3;
parameter cx1 float range from 0 to 0.4;
parameter cx2 float range from 0.2 to 0.6;
parameter cx3 float range from 0.5 to 1.0;
parameter cy1 float range from -0.2 to 0.2;
parameter cy2 float range from -0.2 to 0.2;
parameter cy3 float range from -0.2 to 0.2;

Figure 3: An airfoil geometry (left) and parameter definition (right).

thickness in throughout the airfoil geometries.

XFOIL [28] is then used to determine the velocity and pressure distributions over all the valid airfoil geometries. The velocity distribution obtained from XFOIL is used in the transition prediction module. The correlation-based γ-Re_θ transition prediction model of Langtry and Menter [29], which takes into account the effects of free-stream turbulence intensity and pressure gradient to predict the location of onset of transition, has been used in the current study. To model the intermittent transition zone, the intermittency factor of Narasimha [30] has been used in conjunction with the Linear Combination Model approach described in [31]. An accurate estimation of the total drag is thus obtained from this process.

In this use case, we aim to achieve three objectives:

- To *maximize* the lift-to-drag ratio at a given angle of attack
- To *maximize* the laminar flow in the upper surface
- And to *maximize* the laminar flow in the lower surface

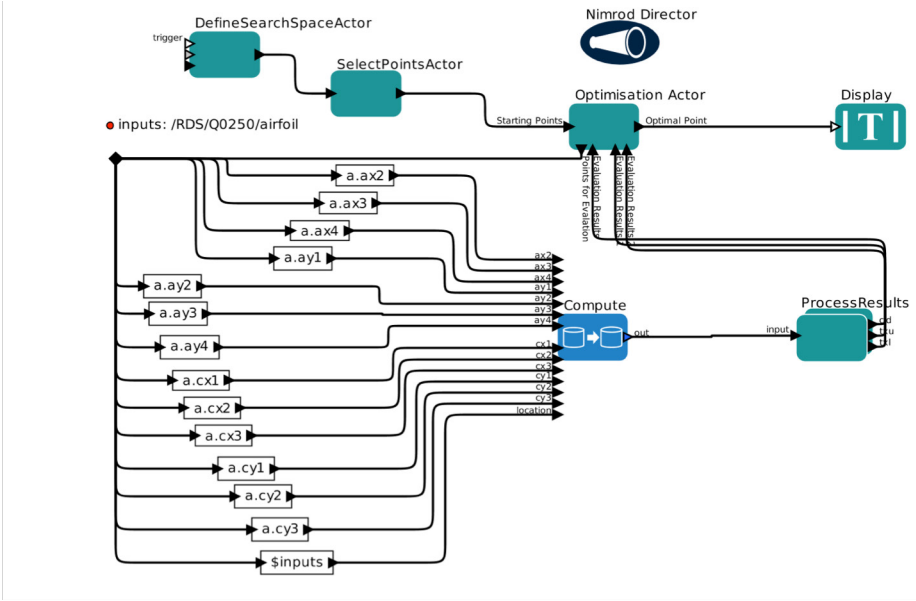


Figure 4: Optimisation workflow

The above optimisation process is implemented as a workflow shown in Figure 4. The *DefineSearchSpace* actor first initiates the optimisation by defining the domain of the search, which is shown in the left part of Figure 4. This search space is specified by the coordinates of the Bezier control points; thickness curve is represented by ax, ay variables while cx, cy variables represent camber curve. The starting points will be selected from this domain by the *SelectPointsActor*, and will be sent to the *Optimisation Actor*. As this workflow involves three objective functions, the *Optimisation Actor* is configured to have 3 objective ports. The workflow executes with both Tabu search and Ganesh implementation. The optimisation actor generates set of points that are sent for evaluation. In this workflow, Bezier curve generation, Xfoil and transition zone estimation are combined into the *Compute* actor, which calculates:

- *cld*: lift-to-drag ratio
- *txu:* x-coordinate of transition point in upper surface

- *txl:* x-coordinate of transition point in lower surface

Since the default option of both Ganesh and MOTS2 are minimization, the values of these results are reversed before they are passed back to the *Optimisation Actor*. This actor decides the next iterations based on these evaluations. The cycle stops when convergence criteria specified in the optimisation actor are met.

5 Results and Discussion

For this experiment, we set the workflow to the typical flight conditions experienced in unmanned aerial vehicles (UAVs): a Reynolds number of $3x10^6$ and Mach number of 0.14 at zero angle of attack. We then execute the workflow with the two newly available algorithms in Nimrod/OK: MOTS2 and Ganesh GA. Both algorithms are started with random starting points and limited to 60,000 evaluations of objective functions in order to compare between them. This corresponds to the setting of 500 generations with 120 individuals in GA and 60,000 iterations in MOTS2. Other configuration settings of both algorithms are left to default.

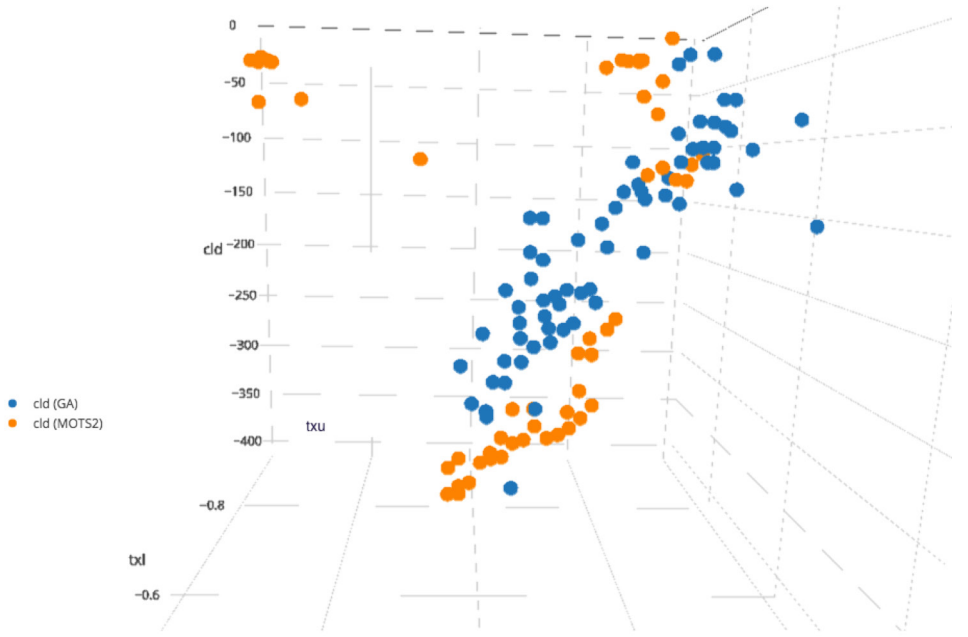


Figure 5: 3D Pareto set.

Figure 5 shows the Pareto set designs of the three-objective optimisation described in the previous section; the plot is available at *https://plot.ly/~hoangnguyen177/9.embed*. As from the figure, both GA and MOTS2 produce similar trends in terms of trade-off points. However, we found that MOTS2 outperforms GA when considering either of these objective sets. When the objective is to maximize both cld and txu, the lift-to-drag ratio reaches maximum when the transition location is between 0.5 and 0.85 of the chord length in both algorithms. MOTS2 and GA show quite different results in the lower surface. While MOTS2 produces highest lift-to-drag ratio when the transition location stays within 0.8 and 0.85 of the chord length, this value generated by GA reaches maximum when the location is between 0.5 and 0.85 of the total chord length. Both algorithms produce similar results when the location of transition is closer to either the leading or the trailing edge of the airfoil. It can

also be seen from the plots that the transition location has little effect on the lift-to-drag ratio, when it is closer to either of the two extremes.

From this experiment, we can conclude that the best designs in terms of high lift-to-drag ratios require transition location to be in the region of 0.5 to 0.8 of the chord at the flight conditions considered during this experiment.

6 Conclusion

The paper has presented the latest development in scientific workflow to enable multi-objective optimisation. This includes an extensible framework and two well-known multi-objective optimisation algorithms. We then use the newly developed capability to solve a state-of-the-art airfoil shape optimisation problem with three objective functions: maximization of lift-to-drag ratio, maximization of laminar flow in upper and lower surface. In this process we include a transition prediction model to predict the onset transition location and the distribution of skin friction through the transition zone for a more accurate estimation of total drag. We have demonstrated the flexibility and convenience offered to the user when optimisation studies are managed through workflow framework, such as Nimrod/OK.

While there is not much work from the research community to formulate and solve multi-objective optimisation problems as workflows, there are several commercial software packages with this feature. Examples of these tools are OPTIMUS, modelFrontier, ModelCentre and Matlab Simulink. Amongst these tools, our work is similar to ModelCentre in which the optimisation process is explicitly expressed in the workflow via iterations. Other tools only allow users to specify the engineering processes that evaluate objective functions; optimisation is performed outside the workflow.

There are several improvements we would like to make to the current work. First, we would like to integrate more optimisation algorithms into Nimrod/OK. Second, we would like to extend the workflow to 3D design applications of each optimal solution in the Pareto front. This should be straightforward as adding new components into a workflow is relatively simple. And third, we would like to include other objectives related to structural, electro-magnetic, aero-acoustics and other disciplines in order to develop workflows for multi-objective, multidisciplinary optimisation processes.

References

[1] T. Kipouros, D. M. Jaeggi, W. N. Dawes, G. T. Parks, a. M. Savill, and P. J. Clarkson, “Biobjective Design Optimization for Axial Compressors Using Tabu Search,” *AIAA Journal*, vol. 46, no. 3, pp. 701–711, 2008.

[2] M. R. Gary and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness.* New York: WH Freeman and Company, 1979.

[3] D. F. Jones, S. K. Mirrazavi, and M. Tamiz, “Multi-objective meta-heuristics: An overview of the current state-of-the-art,” *European Journal of Operational Research*, vol. 137, no. 1, pp. 1–9, 2002.

[4] B. Ludascher, C. Berkley, M. Jones, and E. A. Lee, “Scientific Workflow Management and the Kepler System,” *Concurrency and Computation Practice Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[5] D. Abramson, B. Bethwaite, C. Enticott, S. Garic, T. Peachey, A. Michailova, and S. Amirriazi, “Embedding optimization in computational science workflows,” *Journal of Computational Science*, vol. 1, no. 1, pp. 41–47, May 2010.

[6] F. Xu, M. H. Eres, F. Tao, and S. J. Cox, “Workflow Support for Advanced Grid-Enabled Computing,” in *Proceedings of the UK e-Science All Hands Meeting*, 2004, pp. 430–437.

[7] T. Crick, P. Dunning, H. Kim, and J. Padget, “Engineering design optimization using services and workflows,” *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 367, no. 1898, pp. 2741–2751, 2009.

[8] D. M. Jaeggi, G. T. Parks, T. Kipouros, and P. J. Clarkson, "The development of a multi-objective Tabu Search algorithm for continuous optimisation problems," *European Journal of Operational Research*, vol. 185, no. 3, pp. 1192–1212, 2008.
[9] E. Zitzler, K. Deb, and L. Thiele, "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results," *Evolutionary Computation*, vol. 8, no. 2, pp. 173–195, 2013.
[10] X.-S. Yang, *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.
[11] J. J. Durillo, A. J. Nebro, F. Luna, B. Dorronsoro, and E. Alba, "jMetal : a Java framework for developing multi-objective optimization metaheuristics," 2006.
[12] I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Information Sciences*, vol. 237, pp. 82–117, 2013.
[13] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4J – A Modular Framework for Meta-heuristic Optimization," in *Proceedings of the Genetic and Evolutionary Computing Conference*, 2011, pp. 1723--1730.
[14] S. Wagner and M. Affenzeller, "HeuristicLab: A Generic and Extensible Optimization Environment," in *Adaptive and Natural Computing Algorithms*, Springer, 2005, pp. 538–541.
[15] Y. H. Lim, J. Tana, and D. Abramsonb, "Solving Optimization Problems in Nimrod/OK using a Genetic Algorithm," in *Proceedings of the International Conference on Computational Science, ICCS 2012*, 2012, vol. 9, pp. 1647–1656.
[16] T. Wall, "Java Native Access." [Online]. Available: https://github.com/java-native-access/jna. [Accessed: 30-Jan-2017].
[17] J. Oliver, T. Kipouros, and A. M. Savill, "A Self-adaptive Genetic Algorithm Applied to Multi-Objective Optimization of an Airfoil," *Advances in Intelligent Systems and Computing*, vol. 227, pp. 261–276, 2013.
[18] T. Kipouros, T. Peachey, D. Abramson, and A. M. Savill, "Enhancing and Developing the Practical Optimisation Capabilities and Intelligence of Automatic Design Software," in *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference 2012*, 2012, no. April.
[19] D. Bushnell, "Aircraft drag reduction--a review," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 217, no. October 2015, pp. 1–18, 2003.
[20] M. Jahanmiri, "Aircraft Drag Reduction: An Overview," 2011.
[21] A. Jameson and S. Kim, "Reduction of the Adjoint Gradient Formula for Aerodynamic Shape Optimization Problems," *AIAA Journal*, vol. 41, no. 11, pp. 2114–2129, 2003.
[22] O. G. Amoignon, J. O. Pralits, A. Hanifi, M. Berggren, and D. S. Henningson, "Shape optimization for delay of laminar turbulent transition," *AIAA Journal*, vol. 44, no. 5, pp. 1009–1024, 2006.
[23] X. Chen and R. K. Agarwal, "Shape optimization of airfoils in transonic flow using a multi-objective genetic algorithm," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 228, no. 9, pp. 1654–1667, 2013.
[24] P. R. Viswanath, "Aircraft viscous drag reduction using riblets," *Progress in Aerospace Sciences*, vol. 38, no. 6–7, pp. 571–600, 2002.
[25] A. Dovgal, V. Y. Levchenko, and V. Timopeev, "Boundary layer control by a local heating of the wall," *Laminar-Turbulent Transition (Springer Berlin Heidelberg)*, pp. 113–121, 1990.
[26] A. Hani, J. Pralitsy, O. Amoignon, and M. Chevalier, "Laminar Flow Control and Aerodynamic Shape Optimization," in *KATnet II Conference on Key Aerodynamic Technologies*, 2009.
[27] C. Wauquiez, *Shape optimization of low speed airfoils using Matlab and Automatic Differentiation*. VDM Publishing, 2009.
[28] M. Drela, "XFOIL: An Analysis and Design System for Low Reynolds Number Airfoils," in *Low Reynolds Number Aerodynamics*, 1989, pp. 1–12.
[29] R. B. Langtry and F. R. Menter, "Correlation-based transition modeling for unstructured parallelized computational fluid dynamics codes," *AIAA Journal*, vol. 47, no. 12, pp. 2894–2906, 2009.
[30] R. Narasimha, "On the distribution of intermittency in the transition region of a boundary layer," *Journal of the Aeronautical Sciences*, vol. 24, no. 9, pp. 711--712, 1957.
[31] R. Narasimha and J. Dey, "Transition-zone models for 2-dimensional boundary layers: A review," *Sadhana*, vol. 14, no. 2, pp. 93–120, 1989.