Introducing Kimeme, a Novel Platform for Multi-disciplinary Multi-objective Optimization

Giovanni Iacca(✉) and Ernesto Mininno

Cyber Dyne S.r.l., Via Scipione Crisanzio 119, 70123 Bari, Italy
{giovanni.iacca,ernesto.mininno}@cyberdyne.it

Abstract. Optimization processes are an essential element in many practical applications, such as in engineering, chemistry, logistic, finance, etc. To fill the knowledge gap between practitioners and optimization experts, we developed Kimeme, a new flexible platform for multi-disciplinary optimization. A peculiar feature of Kimeme is that it can be used both for problem and algorithm design. It includes a rich graphical environment, a comprehensive set of post-processing tools, and an open-source library of state-of-the-art single and multi-objective optimization algorithms. In a memetic fashion, algorithms are decomposed into operators, so that users can easily create new optimization methods, just combining built-in operators or creating new ones. Similarly, the optimization process is described according to a data-flow logic, so that it can be seamlessly integrated with external software such as Computed Aided Design & Engineering (CAD/CAE) packages, Matlab, spreadsheets, etc. Finally, Kimeme provides a native distributed computing framework, which allows parallel computations on clusters and heterogeneous LANs. Case studies from industry show that Kimeme can be effortlessly applied to industrial optimization problems, producing robust results also in comparison with other platforms on the market.

Keywords: Multi-disciplinary optimization · Software design · Graphical interface · Distributed computing

1 Introduction

Over the past two decades, Computational Intelligence Optimization (CIO) has become a popular topic among computer scientists and practitioners. The reason for this success is manifold. First of all, industrial and societal problems have become (and still are becoming) ever more challenging, thus requiring robust solvers and algorithms. Despite the advancements in exact methods, e.g. based on classical mathematical optimization, and related tools, such as CPLEX [1], which can efficiently solve many classes of optimization problems, there are still problems whose scale and complexity may hinder their use, including examples of large-scale problems, dynamic problems, problems affected by noise, or black-box problems for which a mathematical formulation is not even available [2,3]. In all these cases, where strong guarantees are not really needed (or feasible) but best-effort "good" solutions are enough, heuristic methods (or "meta-heuristics"), such as those offered by CIO (Evolutionary Algorithms, Swarm Intelligence, etc.), are often the only effective solution [4].

Another reason for the success of CIO is that these methods are based on very little assumptions (or none at all) on the problem at hand. These methods are, in fact, black-box, so that virtually any input/output system (i.e. a system where inputs-the problem design variables-are mapped to one or more outputs-the problem metrics to minimize or maximize, or "fitness" in the evolutionary jargon) can be optimized by using them. This property is especially useful for example in many engineering, networking, or logistic problems where an explicit, closed-form mapping between inputs and outputs is not available but is often the output of a domain-specific simulator.

Other reasons for the success of CIO are the fact that meta-heuristics can usually be applied to various problems with minimum coding/engineering effort, and, finally, the fact they are largely available in the literature. A plethora of methods exist nowadays, with different properties of robustness and self-adaptation; still, the family of these algorithms is growing and every year the state-of-the-art in optimization is pushed forward.

Table 1. Some of the most popular optimization software tools. The column "Customizable" indicates if the software allows to implement new optimization algorithms or customize existing ones.

Software	GUI	MOO	Open-source	Customizable	License
HEEDS MDO [5]	Yes	Yes	No	No	Commercial
HyperStudy [6]	Yes	Yes	No	Yes	Commercial
Isight [7]	Yes	Yes	No	No	Commercial
LIONsolver [8]	Yes	Yes	No	No	Commercial
modeFRONTIER [9]	Yes	Yes	No	No	Commercial
MOPS [10]	Yes	Yes	No	No	Commercial
Nexus [11]	Yes	Yes	No	No	Commercial
OpenMDAO [12]	No	Yes	Yes	Yes	Free
Optimus [13]	Yes	Yes	No	No	Commercial
OptiY [14]	Yes	Yes	No	No	Commercial
SmartDO [15]	Yes	Yes	No	No	Commercial
Xtreme [16]	Yes	Yes	No	No	Commercial
μGP [17]	No	Yes	Yes	Yes	Free
Kimeme [18]	Yes	Yes	No	Yes	Commercial

As a consequence of this trend, several software packages have been developed in the last years, which provide off-the-shelf algorithms for solving multi-disciplinary optimization problems. Among these, especially those tools designed for solving multi-objective optimization problems, i.e. problems where multiple conflicting criteria have to be optimized [19], have gained an increasing success and popularity. This is indeed a class of problems that arises in several domains, such as engineering [20,21] or finance [22], and which is therefore extremely relevant in practical problems.

A short list of such tools is reported in Table 1, where GUI and MOO indicate, respectively, if the software has a Graphical User Interface and if it allows for multi-objective optimization[1]. We also indicate if each tool is open-source and if it allows for customization of the optimization algorithms. We should note that we included in the table only multi-disciplinary optimization tools that are specifically based on CIO methods, while we excluded software based on classic techniques for convex optimization, integer linear/non-linear programming and methods addressing combinatorial optimization only. We also excluded those technical software products that are not devoted specifically to optimization but still may include optimization methods, such as Matlab (which provides the Optimization Toolbox), and other CAD/CAE or multi-physics software, as well as multi-disciplinary tools that provide (as an extra feature) one or more, often domain-specific, optimization techniques (see for instance AVL CAMEO [23]).

Clearly, different commercial and open-source free platforms are characterized by different features in terms of usability, openness, modularity, scalability, easiness of interfacing with external packages, etc. Nevertheless, many of these products share similar principles and patterns. For instance, among commercial software, many tools allow the user to describe the optimization problem graphically (with a data-flow or process-flow approach); typically, they provide a variably rich toolbox for post-processing, and a relatively small set of off-the-shelf optimization algorithms (closed-source, either legacy or from the literature) with limited possibility for parameter tuning or other algorithm modifications. Instead, open-source tools generally lack complete graphical interfaces or advanced post-processing features, thus resulting more difficult to use, at least for practitioners interested in ready-to-use tools; also, they require in general some knowledge about optimization and programming, for instance for writing scripts or markup language files to interface with third-party software; however, due to their openness, these platforms allow a higher level of flexibility (e.g. expert users can develop their own algorithms or modify existing ones relatively easily).

From this short summary we can note that, as a general trend, commercial and open-source optimization tools are rather far apart in terms of usability and flexibility (but this distinction, arguably, affects all technical software): on one hand, commercial tools prefer usability over flexibility, as they typically address the needs of users who are not necessarily specialized in programming and optimization (such as mechanical engineers, designers, logistic experts, and other corporate technical figures for which optimization is simply a tool).

[1] A more complete list is available at: http://en.wikipedia.org/wiki/List_of_optimization_software.

On the other hand, open-source software usually look at academic users, such as mathematicians and optimization scholars who are, in general, more interested in designing novel algorithms and testing them on benchmark functions, rather than applying existing algorithms to specific real-world problems. At the time being, it is hard to find tools that bridge this gap between these two worlds, providing at the same time the robustness and ease of proprietary software and the flexibility of open-source platforms.

Motivated by this idea, we developed a novel commercial multi-disciplinary optimization platform, Kimeme. Similar to other commercial platforms, Kimeme provides, off-the-shelf, a rich graphical interface including the possibility of describing the optimization problem graphically, a large set of post-processing tools, such as plots and statistic analysis, and several CIO methods for both single and multi-objective optimization. Moreover, it includes a native distributed computing system, which allows for a seamless parallelization of the solution evaluations on a local network. However, differently from most commercial software, Kimeme adds the possibility of modifying the code of the existing algorithms, reusing it to design novel algorithms, or even integrating algorithms designed from scratch. In this way, Kimeme tries to bridge the gap between academy and industry, in such a way that ever more powerful algorithms proposed in the specialized literature on optimization can be easily implemented and made immediately available to practitioners who may be willing to apply them to complex real-world problems.

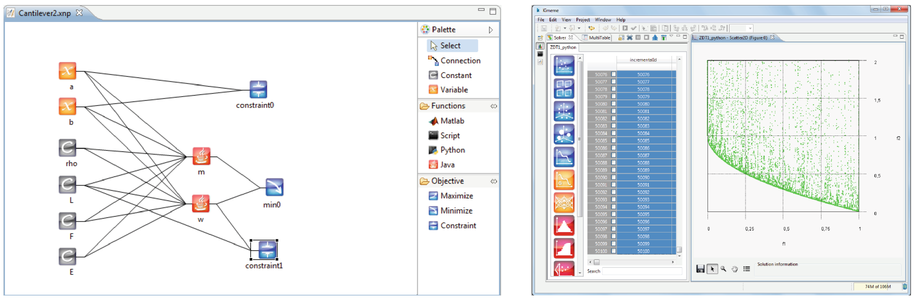
In this paper we present the main features of Kimeme, its architecture and the way algorithms are implemented. First, we describe in Sects. 2.1 and 2.2, respectively, the problem and algorithm design environments, while in Sect. 2.3 we introduce briefly some of the post-processing tools available in the platform. Section 2.4 presents the distributed computing framework integrated in Kimeme, which leverages multiple computers to solve computationally expensive problems. Section 3 presents two case studies to exemplify some possible applications of Kimeme. Finally, in Sect. 4 we give the conclusions of this work.

2 Architecture of Kimeme

The main feature of Kimeme is a rich GUI that assists the user in all the steps of the optimization process, from the problem definition to the results post-processing. In this section, we describe the main components of the Kimeme GUI, namely the problem and algorithm design and the post-processing tools. We also introduce another important feature of the platform, that is its native distributed computing infrastructure.

2.1 Problem Design

In Kimeme, every problem is internally represented as an *evaluation tree*, i.e. an execution tree where each node concurs (in a data-flow logic) to the evaluation of a single solution to the problem at hand. As shown in Fig. 1.a, the tree can



(a) (b)

Fig. 1. Some screenshots of Kimeme: (a) project design view; (b) solution table view, including a 2D scatter plot (see also Sect. 2.3).

be created visually, simply adding nodes from a palette. The palette currently includes several nodes for defining design variables, constraints, objectives, constants, as well interface nodes to external software, such as Matlab, Python or Java code, Bash or DOS scripts, etc.

2.2 Algorithm Design

A key element of Kimeme is the possibility, for the user, to modify existing algorithms provided off-the-shelf by the platform, or implementing new ones. A set of state-of-the-art optimization algorithms is available, both for single-objective problems (such as Differential Evolution (DE) [24], Evolution Strategies (ES) [25], Self-adapting Differential Evolution (jDE) [26], and Nelder-Mead Simplex [27]) and multi-objective problems, including Non-dominated Sorting Genetic Algorithm 2 (NSGA-2) [28], Multi-objective Particle Swarm Optimization (MOPSO) [29], Strength Pareto Evolutionary Algorithm 2 (SPEA2) [30], Archived Multi-objective Simulated Annealing (AMOSA) [31] and two custom variants of Multi-objective DE (MODE) and ES (MOES).

Kimeme was designed with code re-usability in mind: algorithms are, in fact, fully open-source and structured in such a way that every single component of an algorithm can be reused in another algorithm. Inspired by the modern wave of Memetic Computing [32–36], algorithms are structured in self-contained elements called *operators* (in the memetic jargon, a generalization of a “meme”), each one performing simple operations on the problem solutions. As shown in Fig. 2, the generic algorithm structure in Kimeme consists of a Design of Experiment (DoE), which generates the initial candidate solutions for the problem at hand (i.e., the initial “population”), followed by a sequence of *Start Operators*, *Step Operators* and *Stop Operators*. The first ones are executed, only once, at the beginning of the optimization, to perform various initializations needed for the execution of the algorithm. Step Operators represent the core of the algorithm iterations and are executed, repeatedly, until one or more stop condition is met. Finally, the Stop Operators implement various stop conditions and check at the end of each iteration if any of those conditions is met.

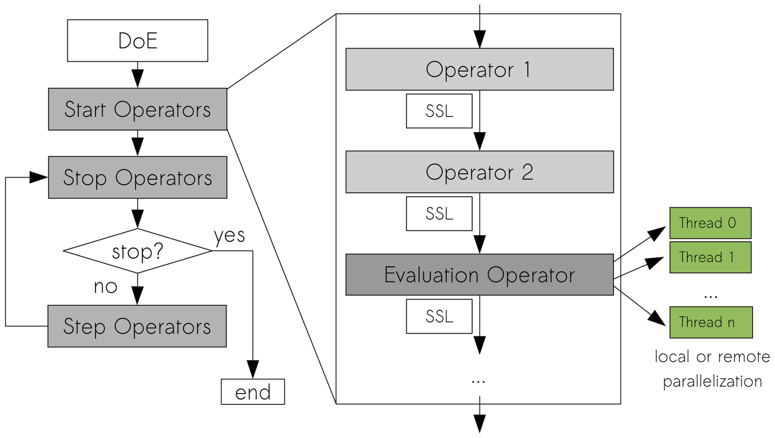


Fig. 2. Generic algorithm flow-chart in Kimeme.

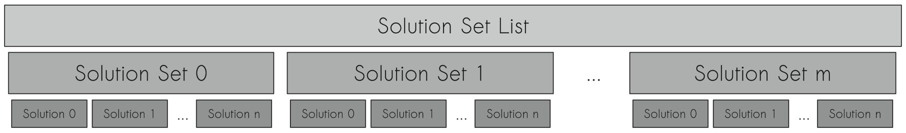


Fig. 3. Structure of the Solution Set List.

The way information is passed among operators is by means of a *Solution Set List* (SSL), i.e. a structured list of candidate solutions (the latter being in turn a structure containing lists of design variables, constraint and fitness values, as well as extra properties of each solution), as depicted in Fig. 3. In this framework, each operator accepts as input a Solution Set List (generated by another operator), and returns as output a new Solution Set List, modified according to its internal operator logics. This structure has two main advantages: on one hand, the use of a structured list of solutions allows an immediate implementation of various modern schemes where multiple populations (composing a "meta-population") co-evolve in parallel, for instance according to an island model, and eventually exchange solutions through some sort of migration mechanisms [37]. Also, this structure is especially useful in multi-objective optimization, where it can be needed to rank the solutions according to their non-domination, thus maintaining different solutions sets, one for each rank.

Kimeme comes with a rich library containing several operators for DoE, crossover, mutation, selection, constraint handling, solution niching and ranking, various utility functions for manipulating the Solution Set List (e.g. merging, cutting, splitting, copying, etc.), different stop conditions, and a number of algorithm-specific operations. Each operator defines one or more specific parameters and additional properties that are needed by the algorithm, and that can also be passed from one operator to another. A special operator, called *Evaluation Operator*, performs the actual evaluation of the incoming SSL, executing multiple instances of the evaluation tree (one for each solution in the SSL) defined in the problem design. Also, this operator is responsible for distributing the computation either locally, on multiple threads, or remotely, through the Kimeme Network (see Sect. 2.4). Apart from the Evaluation Operator, all operators available in Kimeme are open-source and can be dissected, modified and adapted to the users' needs. Operators can be easily implemented in an object-oriented fashion, either in Python or Java (through an Integrated Development Environment within the Kimeme GUI) and interpreted or compiled dynamically at runtime. A complete Java/Python Application Programming Interface (API) is provided to help the users programming operators. The structure of an algorithm (i.e. the sequence of its operators, as well as their parameters and properties) is simply defined by an `xml` file, which can be edited either manually or using the Kimeme GUI. The latter assists the user in various operations such as changing the order of the operators by drag-and-drop, removing or adding new operators, defining their parameters and properties etc. A checker is also provided to automatically verify that there are no errors in the operators' code or inconsistencies in the logics of the algorithm.

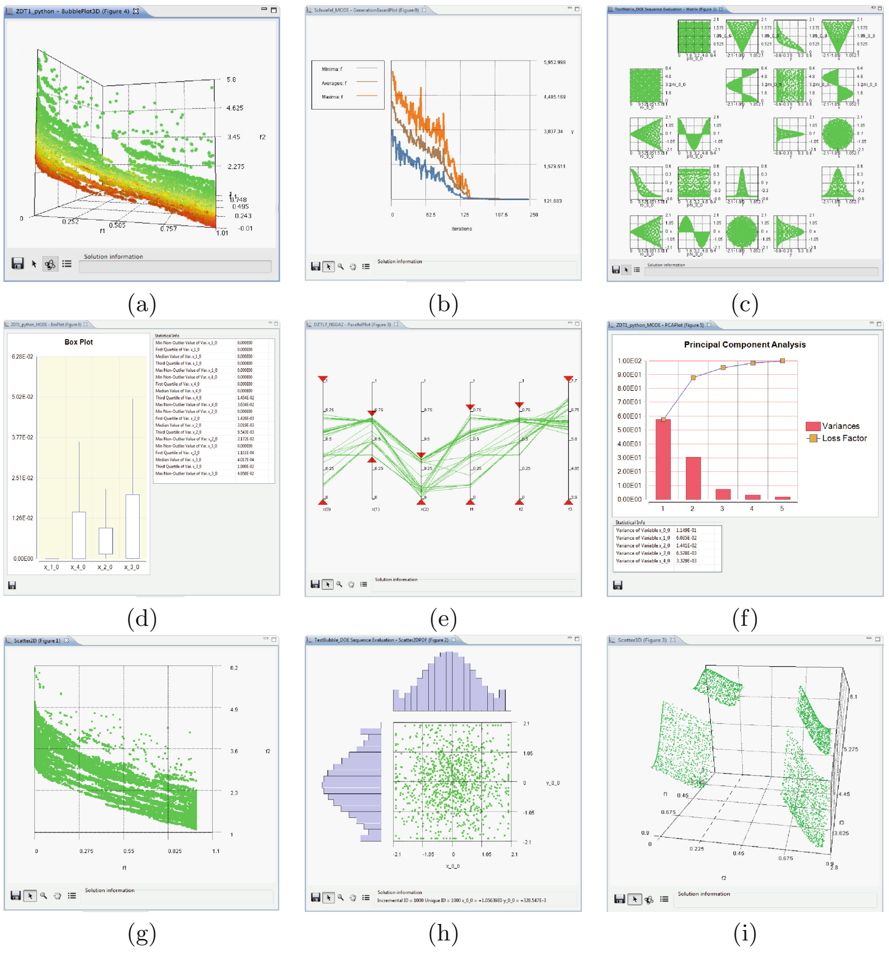
2.3 Post Processing

To analyze the results of the optimization process, Kimeme provides a rich set of plots, statistical analyses and post-processing tools. Some examples of such tools are shown in Fig. 4, including for instance various scatter plots 2D and 3D, and other multi-dimensional visualization plots such as matrix and parallel plots. The typical post-processing use case involves the selection of one or more solutions from the main solution tables (for example those belonging to the Pareto front), and the choice of the desired plots, see Fig. 1.b for an example.

In general, multiple plots can be associated to a single table, providing different levels of information about the solutions generated by the optimization process (for instance their distribution in the search space, the correlation among variables, etc.). The user can easily interact with the plots, for instance zooming in/out, rotating the 3D views, checking a solution and visualizing its details, etc. The plot graphical details (line width, colors, markers, etc.) can also be edited, independently for each plot. Additionally, solutions data and plots can be exported to ASCII, Excel and Matlab files, and various image formats.

2.4 Kimeme Network

We conclude this section with a brief description of the distributed computing feature available in Kimeme, called *Kimeme Network*. Distributed computing is particularly relevant in expensive optimization, for instance in many engineering design problems where each solution evaluation corresponds to a computationally heavy multi-physics simulation. In such scenarios, running an optimization algorithm (which typically requires hundreds of thousands evaluations) on a single computer might introduce a bottleneck in the business processes. To cut design


ZDT1_python - BubblePlot3D (Figure 4)
Solution information
(a)
Schwefel_MODE - GenerationBasedPlot (Figure 0)
Minima: f
Averages: f
Maxima: f
Iterations
(b)
TestMatrix_DOE Sequence Evaluation - Matrix (Figure 1)
(c)
ZDT1_python_MODE - BoxPlot (Figure 0)
Box Plot
(d)
DZTL7_NSGA2 - ParallelPlot (Figure 3)
(e)
ZDT1_python_MODE - PCAPlot (Figure 5)
Principal Component Analysis
Variances
Loss Factor
(f)
Scatter2D (Figure 1)
(g)
TestBubble_DOE Sequence Evaluation - Scatter2DPDF (Figure 2)
(h)
Scatter3D (Figure 3)
(i)

Statistical Info

Min Non-Outlier Value of Var. x_1_0	0.000E00
First Quartile of Var. x_1_0	0.000E00
Median Value of Var. x_1_0	0.000E00
Third Quartile of Var. x_1_0	0.000E00
Max Non-Outlier Value of Var. x_1_0	0.000E00
Min Non-Outlier Value of Var. x_4_0	0.000E00
First Quartile of Var. x_4_0	0.000E00
Median Value of Var. x_4_0	0.000E00
Third Quartile of Var. x_4_0	1.454E-02
Max Non-Outlier Value of Var. x_4_0	3.634E-02
Min Non-Outlier Value of Var. x_2_0	0.000E00
First Quartile of Var. x_2_0	1.426E-03
Median Value of Var. x_2_0	3.019E-03
Third Quartile of Var. x_2_0	9.543E-03
Max Non-Outlier Value of Var. x_2_0	2.172E-02
Min Non-Outlier Value of Var. x_3_0	0.000E00
First Quartile of Var. x_3_0	1.111E-04
Median Value of Var. x_3_0	4.017E-04
Third Quartile of Var. x_3_0	1.990E-02
Max Non-Outlier Value of Var. x_3_0	4.958E-02

Statistical Info

Variance of Variable x_0_0	1.149E-01
Variance of Variable x_1_0	6.035E-02
Variance of Variable x_2_0	1.441E-02
Variance of Variable x_3_0	6.578E-03
Variance of Variable x_4_0	3.329E-03

Fig. 4. Some of the post-processing plots available in Kimeme: (a) 3D bubble plot; (b) generation plot; (c) matrix plot; (d) box plot; (e) parallel plot; (f) PCA plot; (g) 2D scatter plot; (h) 2D scatter plot with probability density function; (i) 3D scatter plot.

cost and time, exploiting the inherent parallel nature of most meta-heuristics, distributed computing is therefore needed.

The main architecture of the Kimeme Network is shown in Fig. 5: a central Java daemon, called *Dispatcher*, receives requests for computation (i.e., individual solution evaluations) from one or more optimization processes instantiated by (different instances of) the Kimeme GUI. The Dispatcher then distributes the computations, according to various scheduling rules, on a list of available computing nodes (that can be modified at runtime), which run another Java daemon called *Worker*. Each Worker defines how many CPUs it has available and how many threads it can launch. Both daemons are platform-independent,

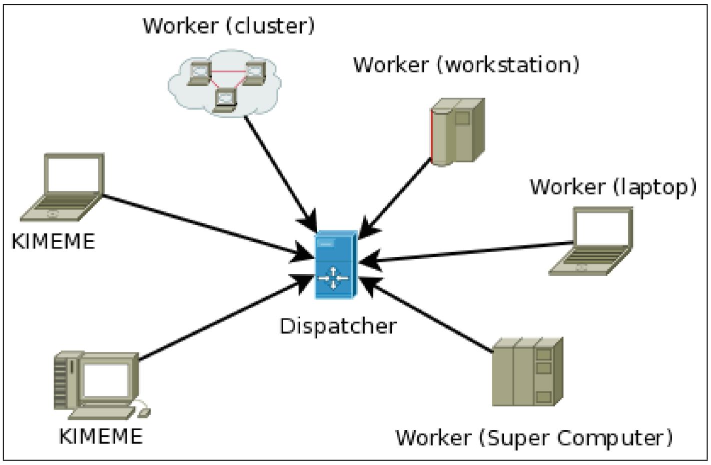


Fig. 5. Kimeme network architecture.

so that it is possible to set up a heterogeneous distributed computing network infrastructure, for instance including Windows/Linux desktops or laptops, next to a cluster and a dedicated high-computing machine.

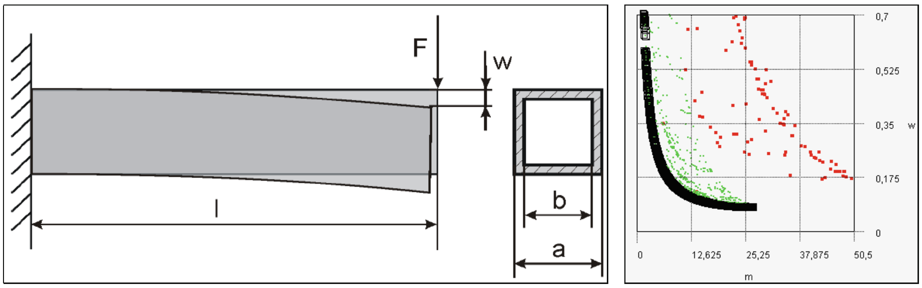
3 Case Studies

We describe here two case studies of application of Kimeme, to exemplify its applicability to engineering and industrial problems. The first case study is a classic structural engineering design problem, shown here for illustration purpose only. The second application is a complex multi-objective metallurgical problem related to the optimization of productivity and CO2 emission of a blast furnace.

3.1 Optimal Design of a Cantilever

This problem consists in designing an aluminum cantilever having a fixed length (l) and a fixed force (F) applied on the free edge, as shown in Fig. 6.a. The goal of the design is to keep the structure light-weighted and rigid or, formally speaking, minimizing both the mass (m) and the deflection (w). Decision variables are the outer (a) and the inner (b) edge lengths. The design has to comply with some dimensional and functional constraints, related to the feasibility of the structure (the outer edge length must be grater than the inner one) and to the deflection-length ratio, which must be less than 10 %. A complete formulation of the problem is available in [38].

Figure 6.b shows the Pareto front generated by the MODE algorithm available in Kimeme. We can observe a nice spread of solutions over the Pareto front, which is covered entirely, and quite evenly.



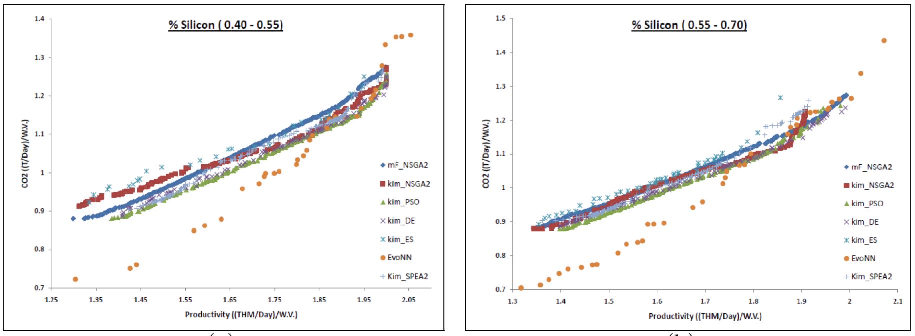
(a) (b)

Fig. 6. Cantilever design problem: (a) problem description and (b) solutions obtained in Kimeme using MODE with the standard parameter setting (population size 100, 250 generations): initial generation (red), last generation (green), Pareto front at the last generation (black dots) (Color figure online).

3.2 Blast Furnace Productivity/CO2 Emission Optimization

This problem consists in minimizing the CO2 emission of an industrial iron blast furnace while simultaneously maximizing its productivity, with a constraint on the Silicon content of the hot metal produced by the furnace. This case study was developed at the Department of Metallurgical and Materials Engineering at the Indian Institute of Technology, and its results have been originally published in a recent paper by Jha et al. [39]. In this work, the authors performed a systematic comparison between various algorithms available in Kimeme and modeFRONTIER [9]. They also compared these algorithms with a custom-developed surrogate-assisted algorithm called Evolutionary Neural Network (EvoNN) [40]. In the study, different levels of Silicon were considered. Here, we report the preliminary results of a smaller subset of algorithms (without tuning, unpublished data) obtained on only two configurations (low and medium, respectively with 0.40–0.55 % and 0.55–0.70 % Si). We refer the interested reader to the work by Jha et al. [39] for further details about the problem formulation and for complete optimization results with tuned algorithms.

Figure 7 shows the Pareto fronts found by five algorithms from Kimeme (NSGA-2, MOPSO, MODE, MOES and SPEA2), one from modeFRONTIER (NSGA-2), and EvoNN, in low and medium Silicon level. It should be noted that, among all the methods shown in the figure, only EvoNN is surrogate-assisted, therefore a fair direct comparison between this method and the other methods is not possible. It can be observed (as also reported in [39]) that all the methods implemented in Kimeme guarantee a rather good solution spread over the Pareto front. Compared with the implementation of NSGA-2 in modeFRONTIER, all the algorithms in Kimeme perform quite well, especially MOPSO and MODE that consistently show good results at all Silicon levels.



(a) (b)

Fig. 7. Pareto fronts obtained on the furnace optimization problem described in [39]: (a) low and (b) medium Silicon level. `mF_NSGA2` indicates the NSGA-2 implementation in modeFRONTIER, while `kim_NSGA2`, `kim_PSO`, `kim_DE`, `kim_ES`, `kim_SPEA2` indicate, respectively, the open implementation of NSGA-2, MOPSO, MODE, MOES and SPEA2 in Kimeme. Lastly, `EvoNN` indicates the surrogate-assisted method proposed in [40]. Courtesy of Nirupam Chakraborti.

4 Conclusions

In this paper we have introduced Kimeme, an innovative platform for designing novel optimization algorithms and applying them to real-world problems. The core idea of Kimeme is to bridge the gap between computer scientists and practitioners, so to foster a mutually beneficial transfer of knowledge between academy and industry. The platform offers a configurable environment for both designing algorithms and solving optimization problems, that can be effortlessly tailored to specific applications from various domains. In addition to that, Kimeme provides a very flexible, scalable and easy-to-use distributed computing infrastructure, that can be used to speed up the optimization process. Moreover, the set of state-of-the-art algorithms provided in Kimeme is broad enough to obtain good results on optimization problems from different domains: the case studies presented here showed that Kimeme could be easily applied to problems from mechanical design and metallurgy, but the platform is naturally applicable to other domains. Further developments are planned in the years to come in order to enrich the platform with new features, optimization algorithms and post-processing capabilities.

References

1. IBM: CPLEX. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/
2. Tenne, Y., Goh, C.K.: Computational Intelligence in Optimization. Springer, Heidelberg (2010)
3. Koziel, S., Yang, X.S.: Computational Optimization, Methods and Algorithms, vol. 356. Springer, Heidelberg (2011)

4. Zelinka, I., Snasel, V., Abraham, A.: Handbook of Optimization: From Classical to Modern Approach, vol. 38. Springer, Heidelberg (2012)
5. Red Cedar Technology: HEEDS® MDO. http://www.redcedartech.com
6. Altair: HyperStudy©. http://www.altairhyperworks.com
7. Dassault Systèmes: Isight©. http://www.3ds.com
8. LIONlab: LIONsolver. http://lionoso.com/
9. ESTECO: modeFRONTIER®. http://www.esteco.com/modefrontier
10. German Aerospace Center, Institute of System Dynamics and Control, AircraftSystems Dynamics: MOPS. http://www.dlr.de/rm/en/desktopdefault.aspx/tabid-3842/6343_read-9099/
11. iChrome: Nexus©. http://ichrome.com/solutions/nexus
12. NASA Glenn Research Center: OpenMDAO. http://openmdao.org/
13. Wilde Analysis Ltd.: Optimus®. http://wildeanalysis.co.uk/fea/software/optimus
14. OptiY GmbH: OptiY©. http://www.optiy.eu/
15. FEA-Opt Technology: SmartDO©. http://www.smartdo.co/
16. Optimal Computing: Xtreme©. http://www.optimalcomputing.be/
17. Sanchez, E., Schillaci, M., Squillero, G.: Evolutionary Optimization: The μGP Toolkit, 1st edn. Springer Publishing Company Inc., Berlin (2011)
18. Cyber Dyne Srl: Kimeme. http://cyberdynesoft.it/
19. Deb, K.: Multi-objective optimization. In: Burke, E.K., Kendall, C. (eds.) Search Methodologies, pp. 403–449. Springer, Heidelberg (2014)
20. Köppen, M., Schaefer, G., Abraham, A.: Intelligent Computational Optimization in Engineering, pp. 300–331. Springer, Heidelberg (2011)
21. Yang, X.S., Koziel, S.: Computational Optimization and Applications in Engineering and Industry, vol. 359. Springer, Heidelberg (2011)
22. Chen, S.H., Wang, P.P.: Computational Intelligence in Economics and Finance. Springer, Heidelberg (2004)
23. AVL: AVL^{TM} CAMEO. https://www.avl.com/cameo
24. Storn, R., Price, K.: Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. J. Global Optim. **11**(TR–95–012), 341–359 (1997)
25. Beyer, H.G.: The Theory of Evolution Strategies. Springer, Heidelberg (2001)
26. Brest, J., Greiner, S., Bošković, B., Mernik, M., Žumer, V.: Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. IEEE Trans. Evol. Comput. **10**(6), 646–657 (2006)
27. Nelder, A., Mead, R.: A simplex method for function optimization. Comput. J. **7**, 308–313 (1965)
28. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
29. Coello Coello, C.A., Lechuga, M.: MOPSO: a proposal for multiple objective particle swarm optimization. In: Proceedings of the 2002 Congress on Evolutionary Computation, 2002. CEC 2002, vol. 2, pp. 1051–1056 (2002)
30. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm (2001)
31. Bandyopadhyay, S., Saha, S., Maulik, U., Deb, K.: A simulated annealing-based multiobjective optimization algorithm: AMOSA. IEEE Trans. Evol. Comput. **12**(3), 269–283 (2008)
32. Iacca, G., Neri, F., Mininno, E., Ong, Y.S., Lim, M.H.: Ockham's razor in memetic computing: three stage optimal memetic exploration. Inf. Sci. **188**, 17–43 (2012)
33. Caraffini, F., Neri, F., Iacca, G., Mol, A.: Parallel memetic structures. Inf. Sci. **227**, 60–82 (2013)

34. Iacca, G., Caraffini, F., Neri, F.: Memory-saving memetic computing for path-following mobile robots. Appl. Soft Comput. **13**(4), 2003–2016 (2013)
35. Neri, F., Cotta, C., Moscato, P.: Handbook of Memetic Algorithms. Studies in Computational Intelligence, vol. 379. Springer, Heidelberg (2011)
36. Caraffini, F., Iacca, G., Neri, F., Mininno, E.: The importance of being structured: a comparative study on multi stage memetic approaches. In: 2012 12th UK Workshop on Computational Intelligence (UKCI), pp. 1–8. IEEE (2012)
37. Mühlenbein, H.: Parallel genetic algorithms, population genetics and combinatorial optimization. In: Becker, J.D., Eisele, I., Mündemann, F.W. (eds.) Parallelism, Learning, Evolution. LNCS, vol. 565, pp. 398–406. Springer, Heidelberg (1991)
38. Cyber Dyne Srl: Kimeme Quick Guide
39. Jha, R., Sen, P.K., Chakraborti, N.: Multi-objective genetic algorithms and genetic programming models for minimizing input carbon rates in a blast furnace compared with a conventional analytic approach. Steel Res. Int. **85**(2), 219–232 (2014)
40. Pettersson, F., Chakraborti, N., Saxén, H.: A genetic algorithms based multi-objective neural net applied to noisy blast furnace data. Appl. Soft Comput. **7**(1), 387–397 (2007)