Chapter 12
Future Trends and Challenges

In this chapter, we give some future trends and challenges of dynamic configuration not only for intelligent optimization algorithm, but also for other algorithms used in the whole life cycle of manufacturing. Firstly, some works related to configuration of intelligent optimization algorithm are introduced. They have similar idea and can be further developed with different kinds of configuration ways. From the perspective of software improvement, we introduce the way of dynamic configuration for other algorithms in manufacturing. From the perspective of hardware improvement, the further development of dynamic configuration on FPGA for lightweight optimization in design, production and maintenance of manufacturing is given. Based on these trends, some challenges in developing dynamic configuration from different angles are listed in this chapter.

12.1 Related Works for Configuration of Intelligent Optimization Algorithm

Intelligent optimization algorithm, also named as meta-heuristic, has been developed for years. Hundreds of evolutionary schemes are presented with population-based iteration and operators. For improving its problem solving capability and make full use of the existing operators, many hybrid mechanisms are emerged. Two of the most famous mechanisms are hyper-heuristic and multi-method search. Hyper-heuristic [1, 2] mainly refers to a heuristic which tries to obtain right methods from a bunch of heuristics for solving a specific problem efficiently. The selection scheme itself, in hyper-heuristic, can be a kind of machine learning techniques or an adapting or turning process. Multi-method search [3, 4] then means to run multiple optimization algorithms simultaneously with population division and combination. The searching process is similar with the parallel configuration of algorithm hybridization mentioned in Chap. 4.

F. Tao et al., *Configurable Intelligent Optimization Algorithm*,
Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-3-319-08840-2_12

(1) Hyper-heuristic

In hyper-heuristic, the goal is to select right algorithms for a specific problem. It is the main difference between hyper-heuristic and meta-heuristic. The selection is based on a bunch of existing algorithms and their performance to some degree. The method for selection is called high-level heuristic, while the algorithms solving the problem in different steps are called low-level heuristics. From the selecting process point of view, it is similar with our dynamic configuration way. More general than dynamic configuration in intelligent optimization algorithm, it tries to choose or combine several kinds of heuristics and machine learning techniques to solve a problem with specific framework [5] step by step. That is to say, it manages a number of heuristics and applies them to different stages of problem-solving. But its basic premise is that the framework or the process for solving a problem is clearly known. It is quite problem-dependent.

Hyper-heuristic is widely studied and applied in different area. With its development, Burke et al. [6, 16] have summarized its main classifications from structure to function. It consists of heuristic selection and heuristic generation, which are similar with our configuration methods in hybridation and in improvement respectively. The main focuses are the design of high-level heuristic based on a known framework, such as [7–9]. In manufacturing, it just applied for combinatorial optimizations such as production scheduling [10, 11], assembly line sequencing [12] and so on. For numerical optimization, parameter optimization and detection problems in manufacturing, few researches have been carried out. Also, the construction of hyper-heuristics largely depends on existing intelligent optimization algorithms [13, 14], as well as our configuration ways.

Broadly, the dynamic configuration ways in intelligent optimization algorithm which encapsulates operators as modules can be seen as a kind of hyper heuristics. That is because it also combines some low level operators (as heuristics) to solve different problems in iteration with some rules. But with generation division especially in intelligent optimization algorithm, dynamic configuration contains not only low level combination of operators, but also high level combination of algorithms. More than that, configuration has different types of schemes for algorithm improvement, hybridation and parallelization. It is independent from problems. Therefore, dynamic configuration is different with hyper-heuristics. It is an extension of component design in intelligent optimization algorithm. All ‘disposable’ and ‘reusable’ [15, 16] operators and algorithms can be reused according to dynamic configuration ways. To some extent, it can also be seen as an extension of hyper-heuristics.

(2) Multi-method Search

Multi-method search is presented by Vrugt et al. [3] who try to overcome the 'no free lunch' theory by applying several algorithms simultaneously to a class of problems. The idea is as well as our configuration method in improvement and hybridation of intelligent optimization algorithm with a simpler style. It is also established based on the population-based iteration rule and some existing algorithms. Different with hyper-heuristics, it sees the operators of an algorithm as an entity and invokes different entities in different sub-population serially. Just as mentioned in the above chapters, if there's one algorithm suitable for the specific problem, it can lead others to searching with a right direction. Now it has been applied for solving both combinatorial and numerical optimization [4, 17].

Nowadays, multi-method search, also known as A Multialgorithm Genetically Adaptive Method (AMALGAM), has been used for optimization in soil and water assessment [18], stochastic inversion in aquifer structure identification [19] and inverse parameter estimation in coupled simulation of surface runoff and soil water flow [20] and so on. But it has not been applied in manufacturing. Moreover, no work has been carried out especially for the design and selection of algorithms during iteration. Therefore, the multi-method search also has long way to go.

In the area of intelligent optimization algorithm, it can be seen that the dynamic configuration contains more ways to reuse existing algorithms fully and widely than the above two mechanisms. Furthermore, just like hyper-heuristic, the idea of configuration can be extensively applied for other algorithms and implemented in other hardware. Here we will take some typical numerical algorithm as an example to show the dynamic configuration ways for other algorithms in manufacturing. Moreover, further dynamic configuration on FPGA will be elaborated as our future work. As a whole, some challenges on the further development of dynamic configuration in the area of manufacturing are given in this chapter.

12.2 Dynamic Configuration for Other Algorithms

Besides optimization, there are a lot of large-scaled complex numerical computing requirements existing not only in manufacturing part design, but also in system and process evaluation, such as large-scaled matrix operations, linear or nonlinear functions and ordinary or partial differential equations. All these calculation process can be completed by a bunch of basic numerical algorithms according to the specific precision demands. Different with intelligent optimization algorithm, these numerical algorithms have different structure and heavily rely on the specific problem. Therefore, the previously mentioned configuration in a uniform iterative process is not adaptable for other numerical algorithms.

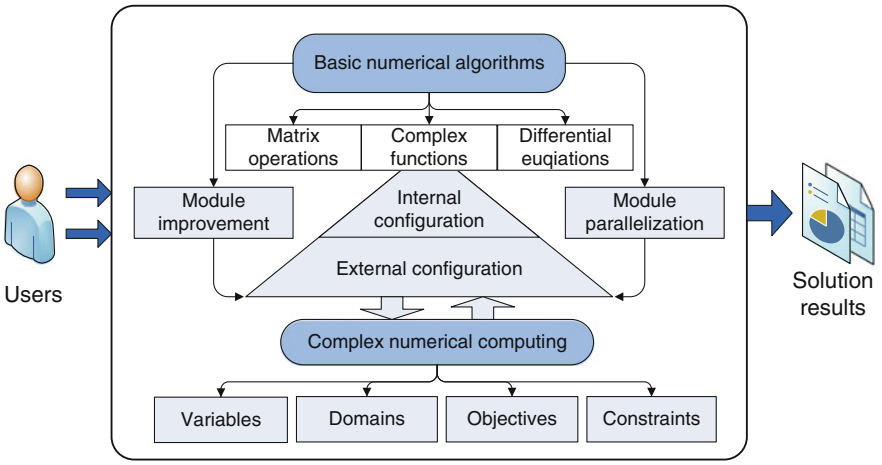

Fig. 12.1 The configuration framework for numerical computing

However, many complex numerical computing modules can be divided into several simple parts with some basic operation. And most numerical problems can be represented as a uniform form with complex variables and constraints. In such standardized form, these classical numerical algorithms can be directly invoked to solve the problem in different stages. Hence the numerical process is generally divisible and can be flexibly configured as well as intelligent optimization algorithm.

Considering three kinds of numerical computing, i.e. matrix operations, complex linear/nonlinear equations and ordinary/partial differential equations, existed generally in design, control and simulation of industrial manufacturing, the configuration framework can be drawn as Fig. 12.1. In this framework, we try to encapsulate basic numerical algorithms as modules and make two level configuration for different kinds of problems, i.e., (1) internal configuration and (2) external configuration.

As mentioned before, complex numerical computing can be divided into several steps. Each step that contains one or more standard numerical computing can also be represented by variables, domains, objectives and constraints. For example, if the numerical computing is linear equations, then it can be represented by a matrix $\mathbf{A}$ and a vector $\mathbf{b}$ (i.e. $\mathbf{Ax} = \mathbf{b}$). If the step needs to solve a partial differential equation, then it can be represented by a partial differential coefficient matrix from which a standard finite difference can be done based on that. In this case, the boundary conditions can be seen as constraints. With such standard representations in each computing stage, basic numerical algorithms which have been encapsulated as module can be configured and connected together.

In configuration, the internal configuration means to change the inner parameter of the module or to invoke other modules internally. For instance, there are many kinds of difference schemes for a specific partial differential equation. With a

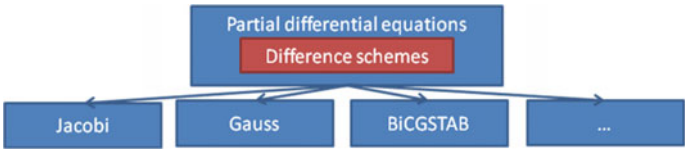

Fig. 12.2 Internal configuration for solving partial differential equations

difference scheme, partial differential equation can be transformed into a group of sparse linear equations. Then Jacobi or Gauss-Seidel method can be used to solve the linear equations. In this case, Jacobi method which is encapsulated into module can be configured with a suitable difference scheme to solve the numerical problem, as shown in Fig. 12.2. Making the transformation of difference scheme also as a module, the configuration above is called internal configuration. By means of multiple basic numerical modules, we could use some heuristic to intelligently select one or more of them connected together for solving a specific numerical problem. Overall, internal configuration in different kinds of numerical computing requires fine grained division of calculating modules. Moreover, all parameters should be adjustable with a standard form to make sure the modules can be correctly invoked.

External configuration, different with the internal one, refers to configure different algorithms in different solving stages for a complex problem. The configured module mainly refers to coarse-grained numerical algorithms. These algorithms can be either a single basic module such like matrix operation, or a combined module configured by the internal configuration as mentioned before. Take the design of aircraft wing for example, the process consists not only the key size design but also the verification of its aerodynamic characteristics. With a bunch of datum, matrix operation may be the first step, the solving of partial differential equations and size decision may be the next. On this occasion, we could configure matrix operation module, Gauss-Seidel and a kind of difference scheme and an intelligent optimization algorithm step by step for solving the whole problem. The outputs of the previous steps are the inputs of the latter ones. Together with internal configuration and external configuration, an integrated numerical algorithm can be generated by a specific heuristic (or algorithm selection rule) and a mass of basic numerical modules.

Moreover, as shown in Fig. 12.1, we could also do module improvement and parallelization during the whole process. Improvement can be easily obtained through flexible parameter interface as mentioned before. Parallelization here consists of both internal and external parallelization.

Internal parallelization means to directly encapsulate basic parallel numerical algorithms as modules. By way of configuring the parallel modules, some new parallel algorithms can be easily produced. As the same as in intelligent optimization algorithm, external parallelization refers to execute different modules simultaneously in different computing nodes. External parallelization is highly dependent on the computing process of a specific problem. For example, we could

separate irrelevant computing module manually to parallel nodes for high time efficiency. We also could apply different numerical algorithms for a single computing stage simultaneously in different computing nodes for high accuracy and stability.

As we know, the establishment of basic numerical modules has been carried out for years. There are already many serial and parallel tools for basic numerical computing, such as matrix operations and solver of linear equations. Basic linear algebra subprograms (BLAS) [21] and Parallel BLAS (PBLAS) [22], LINPACK [23] and PETSc [24] and so on are all famous and have been widely used in different areas. These tools based on different platform and programming language can not be integrated together. If we need to invoke these basic modules, we have to program our problem with the specific programming language and do more changes. With different formation, different numerical modules can not be connected and configured directly. Moreover, there has no recommendation to decide which is the most suitable one for a specific problem.

Therefore, for establishing a configurable platform for wider numerical computing, the uniform encapsulation of existing numerical algorithms is required. In other words, the interface of each module to be configured should be uniformed and full information should be provided for flexible configuration. After that, the most important thing of numerical configuration is the construction of rule base which can be used for algorithm selection according to the problem characteristics and the calculating environments.

12.3 Dynamic Configuration on FPGA

As elaborated in Chaps. 5 and 11, the dynamic configuration of not only intelligent optimization algorithm but also other numerical algorithm can be implemented on FPGA. Focus on the configuration of intelligent optimization algorithm, there are two implementing schemes on FPGA, as shown in Figs. 12.3 and 12.4, (1) operator-based configuration and (2) algorithm-based configuration. It should be noted that an FPGA board can only store limited operators for just one class of problems.

In the first scheme, we could firstly extract initialization part and population updating part into two modules. Operators in iteration can be implemented independently with uniform population-based interfaces. As shown in Fig. 12.3, the connection of these modules can generate a complete intelligent optimization algorithm. *Op1*, *Op2*, *Opx*, *Opy*, *Opz* and *Opn* represent different kinds of operators, such as single-point crossover and mutation in genetic algorithm. The red solid line and the blue chain dotted line represent two kinds of connection ways respectively. With only one connection path between initialization and population updating, a hybrid intelligent optimization algorithm can be generated in which all population is concurrently operated by the corresponding operators on FPGA. In contrast, if there are two or more connection paths, then the population will be

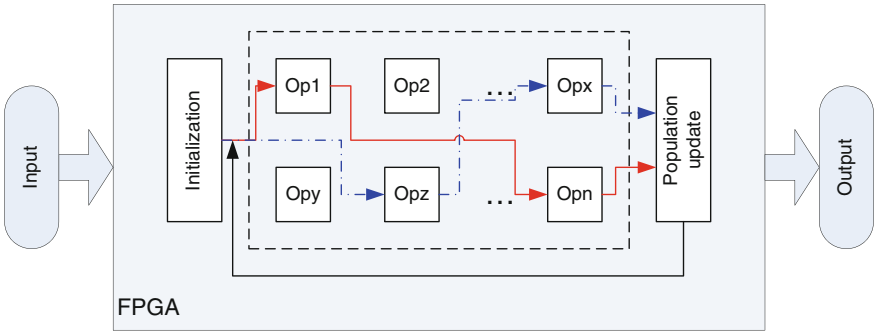

Fig. 12.3 Operator-based configuration on FPGA

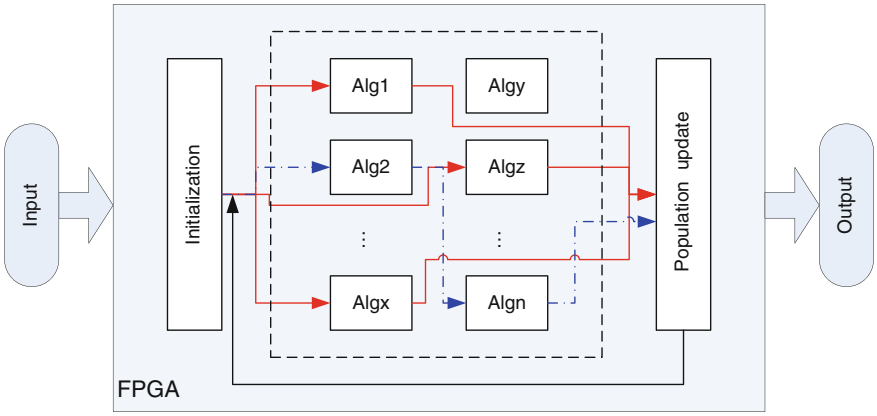

Fig. 12.4 Algorithm-based configuration on FPGA

evenly separated as multiple groups. Each sub-population is executed following the relevant path. A parallel intelligent optimization algorithm with several groups of hybrid operators can be produced. It is the same as the configuration in general cluster environment.

In the second scheme, we could implement the whole operators of a specific algorithm together as one module. For example, if *Alg1* is the classical GA, then the operations in it consist of population selection, crossover and mutation. As shown in Fig. 12.4, we could arbitrarily select different algorithm in parallel with multiple connection paths to form hybrid parallel intelligent optimization algorithm on FPGA. And we could also connect algorithms in serial to generate a single hybrid one. No matter which kinds of connection ways, parallel execution of individuals is ensured on FPGA with high time efficiency.

The implementation of intelligent optimization algorithm on FPGA mainly aims at lightweight and high speed decision in a special environments or devices. One implementation with fixed architecture is just for one specific problem and

can not be reused to others. With the above dynamic configuration, an FPGA board with several operators can then be widely applied to a large amount of problems without reconstruction. Although the implementation is under construction, we believe it will be practical and valuable for many problems ranging from production evaluation to maintenance to realize lightweight high efficiency decision and optimization.

12.4 The Challenges on the Development of Dynamic Configuration

In this book, we elaborated all kinds of dynamic configuration ways for improvement, hybridation and parallelization of intelligent optimization algorithm and their application in manufacturing field. On the basis of various modules, much more efficient algorithms can be generated for different complex problems ranging from part design to system management. A bran-new design conception is introduced for intelligent optimization algorithm. However, the new design method has shifted the difficulty from 'algorithm design' to 'algorithm selection'. Currently, the most important things to establish such dynamic configuration are the construction of algorithm module library and the establishment of recommend rules. The former task has been carried out, as mentioned in Chap. 3, while the latter one as a core part to realize intelligent dynamic configuration is still a big challenge.

Specifically, during the configuration process, designer or engineer have to know the existing operators well enough to select suitable ones for a specific problem. Balance between exploration and exploitation is highly required in configuration. However, to most engineers, it is still not easy to make decision. With less knowledge on different kinds of intelligent optimization algorithms, they have to do large amount of experiments to traverse all these operators and select one or two of them according to the results. It is time consuming. Hence, a rule base is required to provide recommendation in deciding which configuration way is suitable, which operators to select and how to set the parameters in the specific algorithms.

For establishing a rule base, we need to classify the existing problem into different kinds firstly. As introduced in Chap. 2, in the whole life cycle, large amount of numerical function optimization, parameter optimization, detection and classification, combinatorial scheduling and multi-disciplinary optimization exist in not only product design, but also process and system management. However, such classification is far from enough. Each category can further be divided into two kinds in accordance with whether the variables are continuous or discrete. Different variables are represented with different encoding scheme. So that one operator in different encoding scheme is totally different. According to its variables, the problem can be classified as continuous problem, non-sequencing discrete problem

and sequencing discrete problem. For example, traveling salesman problem belongs to sequencing discrete problem, in which its variables are a serial integral number and are different from each other, while traditional task scheduling problem is a kind of non-sequencing discrete problem. Of course in many situations the variables are a blend of continuous and discrete ones. With such classification, when a problem is modeled and submitted, a rule must identify which category it belongs to.

Based on the classification, the second step is to classify the existing operators according to the variable characteristics. For example, the operator of classical ant colony algorithm is suitable especially for sequencing discrete problem, while the operator of traditional particle swarm optimization belongs to continuous operation. With clear operations, this step is easy to implement.

After that, a bunch of heuristics or some learning algorithms, as well as hyper-heuristics, are required for further determining how to select a group of operators and configure them for a specific problem. This is crucial and hard to implement.

On one hand, theoretical verifications of intelligent optimization algorithms are quite less. For a class of problems, we need to take large number of experiments to see whether an operator is suitable. That is very time consuming. If the problem is changed, more tests have to taken to adjust the variation. It can be seen that, the relation between an operator and a class of problems is hard to figure out.

On the other hand, if we have obtained a group of operators, the problems of which configuration approach to use and how to configure them together are also two difficulties for us. From a learning point of view, the construction of recommendation rule also requires large amount of experimental and practical data from a real system.

For overcoming these challenges and establishing a certain level of automatic configuration, a lot of data obtained from a number of experiments based on different sorts of problems and the design of high level machine learning algorithms are both imperative.

In spite of this, dynamic configuration of intelligent optimization algorithm can still be applied with manual control in various kinds of problems not only in manufacturing but also in other fields. Especially by means of parameter-based configuration, operator-based configuration and algorithm-based configuration in both serial and parallel algorithm design, limit operators can produce hundreds of algorithms in different platforms. As a new design mechanism, it can solve wider dynamic problems with uncertainties and complex components through diverse configurations and a bunch of tests.

12.5 Summary

In this chapter, we mainly talked about some related works similar with the dynamic configuration of intelligent optimization algorithm. The similarities and differences between the dynamic configuration and hyper-heuristics and multi-method search are given. Moreover, we showed that the idea of dynamic

configuration can be further developed for other algorithms and on different kinds of hardware platforms. Some future design framework of dynamic configuration for numerical algorithms in manufacturing is drawn. And two kinds of flexible implementations of dynamic configurable intelligent optimization algorithms on FPGA are elaborated.

Currently, all these works are under construction. Each of them can be applied in different area for fast and efficient design and optimization. Further, for realizing automatic configuration, as well as in hyper-heuristics, we summarized some challenges on the development of rule base for dynamic configuration. It is one of the most crucial components which is imperative for engineers and designers, who do not have the comprehensive knowledge of intelligent optimization algorithm, to apply such configuration schemes based on a configurable intelligent optimization algorithm library.

References

1. Burke EK, Hart E, Kendall G, Newall J, Ross P, Schulenburg S (2003) Hyper-heuristics: an emerging direction in modern search technology. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. Kluwer Academic Publishers, Boston
2. Ross P (2005) Hyper-heuristics, search methodologies. Springer, Berlin
3. Vrugt JA, Robinson BA (2007) Improved evolutionary optimization from genetically adaptive multimethod search. Proc Natl Acad Sci 104(3):708–711
4. Vrugt JA, Robinson BA, Hyman JM (2009) Self-adaptive multimethod search for global optimization in real-parameter spaces. IEEE Trans Evol Comput 13(2):243–259
5. Qu R, Burke EK (2009) Hybridisations within a graph based hyper-heuristic framework for university timetabling problems. J Oper Res Soc 60:1273–1285
6. Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E, Woodward JR (2010) A classification of hyper-heuristic approaches. In: Gendreau M, Potvin JY (eds) Handbook of metaheuristics. Springer, New York, pp 449–468
7. Hart E, Ross P, Nelson JAD (1998) Solving a real-world problem using an evolving heuristically driven schedule builder. Evol Comput 6(1):61–80
8. Ochoa G, Qu R, Burke EK (2009) Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. In: Proceedings of the ACM genetic and evolutionary computation conference (GECCO), pp 341–348
9. Burke EK, McCollum B, Meisels A, Petrovic S, Qu R (2007) A graph-based hyper-heuristic for educational timetabling problems. Eur J Oper Res 176:177–192
10. Vazquez-Rodriguez JA, Petrovic S, Salhi A (2007) A combined meta-heuristic with hyper-heuristic approach to the scheduling of the hybrid flow shop with sequence dependent setup times and uniform machines. In: Proceedings of the 3rd multidisciplinary international scheduling conference: theory and applications (MISTA)
11. Rodríguez JAV, Salhi A (2007) A Robust meta-hyper-heuristic approach to hybrid flow-shop scheduling. In: Evolutionary scheduling. Springer, Berlin, pp 125–142
12. Cano-Belmán J, Ríos-Mercado RZ, Bautista J (2010) A scatter search based hyper-heuristic for sequencing a mixed-model assembly line. J Heuristics 16(6):749–770
13. Bai R, Burke EK, Kendall G (2007) Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation. J Oper Res Soc 59(10):1387–1397

14. Burke EK, Kendall G, Soubeiga E (2003) A tabu-search hyperheuristic for timetabling and rostering. J Heuristics 9(6):451–470
15. Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E, Woodward J (2009) Exploring hyper-heuristic methodologies with genetic programming. In: Mumford C, Jain L (eds) Collaborative computational intelligence. Springer, Berlin, pp 177–201
16. Burke EK, Hyde M, Kendall G, Woodward J (2010) A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics. IEEE Trans Evol Comput 14(6):942–958
17. Vrugt JA, Robinson BA, Hyman J (2008) A universal multimethod search strategy for computationally efficient global optimization. Geological Society of America (GSA), New York, pp 28–31
18. Zhang X, Srinivasan R, Liew MV (2010) On the use of multi-algorithm, genetically adaptive multi-objective method for multi-site calibration of the SWAT model. Hydrol Process 24(8):955–969
19. Harp DR, Dai Z, Wolfsberg AV, Vrugt JA, Robinson BA, Vesselinov VV (2008) Aquifer structure identification using stochastic inversion. Geophys Res Lett 35(8):L08404
20. Köhne JM, Wöhling T, Pot V, Benoit P, Leguédois S, Bissonnais YL, Šimůnek J (2011) Coupled simulation of surface runoff and soil water flow using multi-objective parameter estimation. J Hydrol 403(1):141–156
21. Lawson CL, Hanson RJ, Kincaid D, Krogh FT (1979) Basic linear algebra subprograms for FORTRAN usage. ACM Trans Math Softw 5:308–323 (Algorithm 539)
22. Choi J, Dongarra J, Ostrouchov S, Petitet A, Walker D, Whaley RC (1996) A proposal for a set of parallel basic linear algebra subprograms. In: Applied parallel computing computations in physics, chemistry and engineering science. Springer, Berlin, pp 107–114
23. Dongarra JJ (ed) (1979) LINPACK users' guide, vol 8. Siam, Philadelphia
24. Balay S, Gropp WD, McInnes LC, Smith BF (1996) PETSc 2.0 users manual. Mathematics and computer science division (UC-405), Argonne National Laboratory