THE UNIVERSITY *of* LIVERPOOL

Multi-objective Optimisation using Learning Automata and its Applications in Power Systems

Thesis submitted in accordance with the
requirements of the University of Liverpool
for the degree of Doctor of Philosophy

in

Electrical Engineering and Electronics

by

Huilian Liao, B.Sc.(Eng.), M.Sc.(Eng.)

September 2011

Multi-objective Optimisation using Learning Automata and its Applications in Power Systems

by

Huilian Liao

Acknowledgements

I would like to give my heartfelt thanks to my supervisor, Professor Q. H. Wu, whose encouragement, guidance and support enabled me to develop a deep understanding of my work. Without his consistent and illuminating instruction, my research work and my life could not proceed to this stage. The research skill, writing skill and presenting skill he taught me will benefit me throughout my life, as well as his inspiring insight into life philosophy and poetry.

I would like to show my gratitude to Dr. L. Jiang, my second supervisor, for his kind guidance with his knowledge of power systems. I would also like to thank Overseas Research Students Awards Scheme for the financial support it provided for my research work at the University of Liverpool.

I offer my regards and blessings to all of the members of the Intelligence Engineering and Industrial Automation Research Group, the University of Liverpool, especially to Dr. W.H. Tang, Dr. M.S. Li, Dr. D.Y. Shi, Dr. T.Y. Ji, Dr. J. Buse, Mr. L. Wang, and Dr. W.J. Tang. Special thanks also go to my friends, Lesley, Louis, Tony, Nick, and a precious couple Trevor and Ning, for their support and friendship. My thanks also go to the Department of Electrical Engineering and Electronics at the University of Liverpool, for providing the research facilities that made it possible for me to carry out this research.

Last but not least, my thanks go to my beloved family for their loving considerations and great confidence in me through these years.

Abstract

Learning automata are a major branch of machine learning designed to find the optimal action to a learning task in a random environment. Interactions with environment and repetitive learning of a number of individual units, which are independent and structurally simple, enable the learning automata to tackle complex learning problems. Systems built with learning automata have been successfully employed in many difficult learning situations over the years. They have also been investigated in solving optimisation problems. However, the performance of the learning automata in solving complex optimisation problems, such as high-dimensional optimisation problems and multi-objective optimisation problems, has not been fully investigated. Therefore, this thesis is devoted to exploring the potential of learning automata in solving complex optimisation problems. In the thesis, Function Optimisation by Learning Automata (FOLA) and Multi-objective Optimisation by Learning Automata (MOLA) have been developed for single and multi-objective complex optimisation problems respectively.

In FOLA, the search domain of a complex optimisation problem is divided into cells and represented by cell values. Each automaton of FOLA conducts dimensional search actions according to the path values which are calculated based on the cell values situated on the searching path. During the optimisation process, cell values are continuously updated using the values of the automata states, and stored in memory. In this way, the information obtained prior to the current state can be collected and efficiently used. With these approaches, FOLA is able to undertake search in continuous states and achieve accurate solutions efficiently. To fully analyse the performance of FOLA, it has been tested based on twenty-two benchmark functions [1], which represent a wide range of challenging optimisation problems. FOLA has been compared with ten Evolutionary Algorithms (EAs), which are widely used for solving complex optimisation problems nowadays, and four newly-proposed EAs which have been reported to solve the same benchmark functions promisingly in literature. The experimental results have demonstrated the superiority of FOLA over the other EAs for most benchmark functions, in terms of the convergence rate and accuracy of finding optimal solutions. FOLA has shown its capability to solve high-dimensional multi-modal problems. The experiment also shows that FOLA is able to greatly reduce computation time, especially for high-dimensional functions.

Most optimisation problems existing in the real world have more than one objective. These problems aim to find evenly distributed Pareto fronts which are the plots of the objective function values of the optimal solutions [2]. They can be tackled by combining the multiple objectives into one single objective function that can be solved by a single-objective optimisation algorithm. However, this method suffers from the drawback of large computation load, and has difficulty in finding non-convex Pareto fronts. Therefore, it is important to develop alternative optimisers that can be used for complex multi-objective problems. Based on FOLA, MOLA is proposed to solve complex multi-objective optimisation problems. MOLA mainly comprises two processes: the process of searching and the process of learning from neighborhood. The process of searching is carried out through a tournament that is held between Pareto global search and Pareto local search. This tournament can lead to a better trade-off between exploitation and exploration, which is a critical factor in finding the optimal solution. In the process of learning, the relationship of neighborhood among the non-dominated solutions is investigated, as it is believed that useful information that can benefit the search is embeded in neighborhood. Based on the relationship, non-dominated solutions are updated based on their neighbors. Through these processes, MOLA is able to find evenly distributed Pareto fronts for complex optimisation problems. MOLA has been compared with two popular weighted-sum based algorithms, Multi-Objective Genetic Algorithm (MOGA) and Multi-Objective Particle Swarm Optimiser (MOPSO), on four multi-objective benchmark functions that comprise low and high-dimensional models, convex and non-convex models, and continuous and discontinuous models respectively. Besides, MOLA has been also compared with the latest developments of Pareto front-based multi-objective algorithms, Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) and Non-dominated Sorting Genetic Algorithm II (NSGA-II), on the basis of thirteen widely used multi-objective functions [3], which comprise complex Pareto set shapes. The simulation results have shown that MOLA greatly exhibits its superiority over the other algorithms, as it can find accurate and evenly distributed non-dominated solutions, and its Pareto fronts are wider than those obtained by the other algorithms. Besides, MOLA consumes less computation time, whilst finding more accurate non-dominated solutions.

In the thesis, the application of FOLA and MOLA in solving optimal power flow problems of power systems has been investigated. Optimal power flow problems are very important in power system operation and planning, especially economic power system dispatch and voltage stability enhancement problems, which have attracted more and more attention around the world. FOLA has also been applied to solve the power flow problems which concern with fuel cost minimisation, voltage profile improvement and voltage stability enhancement, based on the IEEE 30-bus and IEEE 57-bus systems. FOLA is fully compared with improved Particle Swarm Optimisation (PSO) and Genetic Algorithm (GA). The simulation results have demonstrated that FOLA is able to offer more accurate solutions with shorter computation times, in comparison with the improved PSO and GA, particularly on the IEEE 57-bus system. FOLA is also applied to solve the optimal power flow problems in the power systems where the operation condition varies for a short period time. Although the varying operation condition is considered here, these problems are considered as static problems in a short period of time. In this case, the fluctuating power output will affect the power flow calculation, and it can cause instability which results in severe detriments in the power systems. In this case, an algorithm which can provide security to the power systems is highly demanded. Simulation studies have been carried out among FOLA, the improved PSO and GA, based on the modified IEEE 30-bus and 57-bus systems, which are embedded with time-varying power outputs. The simulation results have demonstrated that FOLA is able to track the changes of the power system configuration more rapidly and accurately than the improved PSO and GA, particularly when voltage stability is involved in the objective function. Besides, FOLA is able to offer more accurate solutions with shorter computation time, in comparison with PSO and GA. FOLA is also compared with two recently-proposed EAs, Comprehensive Learning Particle Swarm Optimiser (CLPSO) and Cooperative Particle Swarm Optimisation (CPSO), based on the IEEE 118-bus system. Advantages of FOLA have been demonstrated by the fact that FOLA reduces the fuel cost greatly and enhances the voltage stability of the power system. Nowadays, wind power is expected to be largely increased in power systems, due to its inexhaustible and nonpolluting merits. However, it brings new challenges to power system operation when wind power is connected to the grid of power systems. The study is undertaken on the modified IEEE 30-bus power system and new England test power system, which are incorporated with fixed-speed and variable-speed wind generators respectively. MOLA has been fully compared with MOEA/D and NSGA-II in solving the multi-objective optimisation problem, which aims to reduce the operational cost and enhance voltage stability simultaneously. The simulation results have demonstrated that MOLA performs better than MOEA/D and NSGA-II, as MOLA can find wider and evenly distributed Pareto fronts, and obtain more accurate Pareto optimal solutions efficiently. Additionally, MOLA consistently finds larger hypervolume and smaller diversity metric than MOEA/D and NSGA-II under different circumstances. MOLA has presented its superiority by finding wider Pareto fronts than MOEA/D and obtaining more accurate solutions than NSGA-II, while using much less function evaluations. MOLA has also been applied to solve the multi-objective optimisation problem in deregulated market, which aims to maximise the social benefit and enhance voltage stability in the IEEE 30-bus power system. MOLA greatly increases the social benefit and improves the voltage stability. It can find wide and evenly distributed Pareto fronts, and obtain accurate Pareto optimal solutions efficiently.

Declaration

The author hereby declares that this thesis is a record of work carried out in the Department of Electrical Engineering and Electronics at the University of Liverpool during the period from October 2008 to September 2011. The thesis is original in content except where otherwise indicated.

Contents

List of Figures

List of Tables

List of Abbreviations and Notations

Chapter 1

Introduction

1.1 Motivations and Objectives

While the mathematics of optimisation has been studied for about a century, the increasing complexity of real-world optimisation problems and the development of computation capability have stimulated new interest in the topic. Classical gradient-based optimisation algorithms have been fully investigated and applied for solving a wide range of current engineering and public service problems [4]. However, they are insufficient when solving the multi-model optimisation problems which are non-differentiable and non-convex and contain many local optima. Furthermore, their convergence is largely dependent on the initial points of search. To solve these problems, researchers began to explore the biological evolution and animal behaviors in the nature, which serves as a fertile source of concepts, principles and mechanisms. Biologically-inspired optimisation algorithms have thrived over the last few decades due to their ability of solving multi-modal problems [5], especially Evolutionary Algorithms (EAs), which present their applicability in solving complex problems. However these algorithms suffer from the drawbacks of redundant computation load and slow convergence rate, *i.e.* they can hardly achieve an accurate solution given limited computation time. These drawbacks mainly result from their population-based search approach, in which there is a high level of randomness and high computational complexity. This has greatly hampered the capability of EAs' application in large-scale optimisation problems, such as optimal power flow problems in power systems, routing problems in telecommunication networks and traffic systems. Further improvement of EAs is limited. This thesis is concerned with an alternative approach to function optimisation, based on learning automata.

Nature has presented the significance of learning ability. For instance, in the wilderness which is full of dangers, each careless action could lead to death, thus creatures have to learn how to select the actions which are suitable to the environment. With this inspiration, the feasibility of using a learning method for problem solving has been explored by researchers [6], such as learning automata methods. Learning automata methods do not belong to the class of EAs which mainly adopt biological evolution concept. A learning automaton is considered as a system which modifies its strategy on basis of its experience by collecting and processing information regarding the environment, in order to achieve the desired goal or the optimal performance in some sense. It is believed that the learning automata, which have the ability of learning and memorization, can be connected in a way which would be suitable for tackling complex learning problems. The learning automata methods have made a significant impact on many areas, such as system control and pattern classification. They have also been applied to resolve optimisation problems. In contrast to EAs, the learning automata methods, despite having a solid theoretical background, are less popularly applied for solving complex optimisation problems. The objective of this thesis is to discover the potential of Learning Automata methods (LA) by developing LA-based optimisation algorithms, which can reduce the unnecessary randomness and large computation load caused by a large number of population adopted by EAs.

Besides the single objective optimisation mentioned above, multi-objective optimisation is with no doubt a very important research topic, due to the multi-objective nature of many practical optimisation problems in the real world. Multi-objective optimisation aims to optimise several performance attributes of the problem simultaneously and obtains a series of non-inferior alternative Pareto optimal solutions. There are two standard methods for treating multi-objective problems. One is to combine the individual objective functions into a single composite function using the weighted-sum method or weighted Tchebycheff method. However, the problem lies with the proper selection of the weights or utility functions to characterize the decision-maker's preferences. In practice, it can be very difficult to precisely and accurately select these weights, even for someone who is familiar with the problem domain. Small perturbations in the weights can sometimes lead to quite different solutions [7]. Another drawback of these approaches is that they greatly increase the computation load, as the algorithm needs to be executed many times in order to obtain a set of Pareto optimal solutions. In addition, these weighted-objective methods are only capable of solving convex Pareto front problems and have a difficulty in solving the multi-objective problems whose Pareto fronts are non-convex [7]. Nonetheless, there is no way to predetermine if a problem is convex or concave in many applications. Instead of using weighted-sum method, an alternative is employing Pareto-front based methods, which apply a population of individuals, and each of them represents one Pareto optimal solution. These approaches are extended from EAs, thus they suffer from the same drawbacks regarding population-based methods, which have been mentioned above. Besides, some of these methods employ non-dominated sorting strategy to rank the individuals of the population, which could further increase the computational complexity. This thesis is to develop an LA-based multi-objective optimisation algorithm, so as to increase the efficiency of the search, through comprehensively making use of the information collected in each function evaluation.

Another objective of the thesis is to apply the developed LA-based algorithms to solve practical problems in power systems. Power system dispatch, environmental concerns and power security are becoming more and more important in terms of power system operation and planning. These problems are non-differential, non-linear, high-dimensional and complicatedly constrained. As it is known that the power system configuration ($e.g.$ power demand, wind power outputs) changes throughout the whole day, which leads to the varying property of the optimisation problem. To a certain extent, the property of these problems is unknown in advance, and it can vary widely as power configuration slightly changes. These features make the problems hard to be resolved. The development of the algorithms which are suitable for this specific area has not been paid enough attention yet, though some efforts have been made by researchers who favor EAs. With the high demand of these problems, the application of EAs is limited in this case, due to their instability and unpredictable computation load. To overcome this problem, this thesis is also concerned with the application of LA-based algorithms in complex power systems.

1.2 Optimisation Algorithms

Optimisation refers to systematically finding the optimal solutions to the optimisation problems to be resolved, under certain constraints. There are mainly two categories of optimisation problems, single-objective and multi-objective optimisation problems. Single-objective optimisation problems aim to finding a single best solution, which is usually the minimal or the maximal evaluation value of the objective function. On the other hand, multi-objective optimisation problems usually have no unique, perfect solution, but a series of non-inferior alternative solutions, Pareto optimal solutions (or called non-dominated solutions), which represent the possible trade-off among conflicting objectives. The Pareto optimal solutions form a Pareto front in the objective space. The optimisation target is to find the set of Pareto optimal solutions which are evenly distributed along the Pareto front [2].

In order to seek the solution which has the best evaluation value of the objective functions, or a desired combination of two or more optimal fitness values of the conflicting objectives, a variety of optimisation algorithms have been developed in the past century. Classical optimisation algorithms have been largely applied for solving a wide range of engineering and public service problems [4]. However, these algorithms cannot solve complex multi-model optimisation problems desirably. Researchers began to explore the nature over the past several decades, and develop various bio-inspired/nature-inspired optimisation techniques, which operate in a rather different way from the classical methods, and allow scientists and engineers to solve optimisation problems where the classical methods are not applicable.

1.2.1 Classical optimisation algorithms

Classical optimisation algorithms, having a solid theoretical background, are analytical and usually make use of the techniques of differential calculus to locate the optimum point. They assume that the function is differentiable twice with respect to the design variables, and the derivatives are continuous. Since some of the practical problems involve objective functions that are not continuous and/or differentiable, the classical optimisation techniques have a limited scope in practical applications. However, classical optimisation algorithms still play an important part in the field of optimisation. For instance, simplex-based method [8] is a popular algorithm for numerically solving linear programming problems [9]. Interior point methods are widely used to solve linear and nonlinear convex optimisation problems [10][11], which have a convex landscape of the objective functions. In order to improve convergence rate, many classical optimisation methods are based on evaluating Hessians and gradients [4], such as Newton's method, Quasi-Newton methods and steepest descent, and so on. The drawback of these methods is that they increase the computational cost of each iteration, due to the computational complexity of evaluating gradients and Hessians.

1.2.2 Evolutionary Algorithms

Since classical optimisation algorithms cannot be used to solve complex multimodel optimisation problems desirably, a great deal of research activities were carried out from the inspiration of nature [12], and since then, a large number of new optimisation algorithms have been developed. These methods have been used to solve various optimisation problems. Meanwhile, these novel optimisation algorithms do not need to run for a large number of times when solving multi-modal problems, which makes their application efficient. Among these novel optimisation algorithms, EAs, which incorporate the major behaviours of a biological evolutionary process and a principle of 'the survival of the fittest' into their algorithmic framework [13], have been investigated comprehensively over the last twenty years and applied widely for problem solving [14][15]. EAs use a population of individuals which evolve through iterative process, in order to search for a desired location in the solution space.

Genetic evolution-based EAs

Genetic evolution-based EAs fall into four major categories: Genetic Algorithm (GA) [16], Evolutionary Programming (EP) [17], Genetic Programming (GP) [18] and Evolutionary Strategy (ES) [19]. These methods share the principles of survival of the fittest.

- The basic concept of GA was first pioneered by John Holland in the 70s [16], and it derives from a metaphor of the evolution process in nature. To be specific, GA refers to a particular class of EAs that uses the techniques inspired by evolutionary biology, such as inheritance, mutation, selection, and crossover [16].

  GA is implemented using a population of strings (called chromosomes), which encode candidate solutions (called individuals) to an optimisation problem, and evolve towards better solutions [20] [21]. Traditionally, the strings were in the form of binary values, but later, other forms, such as real-value coding [22], are also possible [23]. The selection of the type of coding relates to the types of the optimisation problem. General GA includes five basic operations: initialisation, selection, crossover, mutation and termination [16]. During the stage of initialisation, a population of individuals are randomly generated in the solution space. These solutions evolve as generations proceed. In each generation, the fitness value of each individual in the population is evaluated. Individual solutions are selected through a fitness-based process, where the solutions that have better fitness values are often more likely to be reselected. Some selection strategies rate the fitness value of each solution and preferentially select the best solutions. However, most selection methods adopt probability-based selection, in which the probability of reselecting better solutions is large, and at the same time the solutions which have relatively poor fitness values can also be selected, but in a smaller probability.

This strategy helps GA preserve population diversity and prevent premature convergence. The most well-studied selection methods include roulette wheel selection and tournament selection [24]. Through the stage of selection, an intermediate population is obtained and used to reproduce a new population. In this case, multiple individuals are stochastically selected from the current population to perform operators crossover and mutation. Crossover and mutation are regarded as the main causes of the efficiency of genetic algorithms. Crossover allows the method to combine some hopeful schemata, join the information contained in the parent chromosomes, and produce new individuals. With crossover, good results can be obtained with a random matching of the individuals [22], and thus quickly progress towards the optimal regions of the search space. On the other hand, mutation brings the diversity among the population, by changing the values of part of the chromosomes. With the operations of crossover and mutation, a brand new population is generated, and it will be used in the next generation. The algorithm either terminates when a maximum number of generations has been produced, or stops when a satisfactory fitness level for the population has been reached.

GA can be well extended to solve multi-objective optimisation problems. Reference [7] provides an overview and tutorial of GA which is developed specifically for problems with multiple objectives. There are mainly two ways of applying GA in solving multi-objective problems: 1) multiple objectives can be regulated into a single objective, using weighted-sum method or Lagrange method, before applying GA. This method is widely used, such as Weight-Based Genetic Algorithm [25], Random Weighted Genetic Algorithm [26] and Multi-Objective Genetic Algorithm (MOGA) [7]; 2) being a population-based approach, a generic single-objective GA can be modified to find a set of multiple non-dominated solutions in a single run, such as Non-dominated Sorting Genetic Algorithm [27], Dynamic Multi-objective Evolutionary Algorithm [28] and Fast Non-dominated Sorting Genetic Algorithm (NSGA-II) [29].

The domain of GA's utilization is very large. They have been widely studied, experimented and applied in many fields in engineering worlds [21][30][31][32]. A review of their implementations and some application domains can be found in reference [33].

- EP was first proposed as an approach to artificial intelligence by Fogel in 1960 [17]. However, it was not applied with such a success to many numerical and combinatorial optimisation problems until 1990 [34].

  Similar to GA, EP seeks the optimal solution by evolving the population, in which each individual represents a candidate solution to the problem, over a number of generations or iterations. However, unlike other EAs, no recombination operators are applied in EP. The optimisation process can be summarized into two major steps [17]: the solutions locating in the current population are mutated; the next generation is selected from both the mutated and the current solutions. According to survival of the fittest, the filial generation in EP is generated from the fittest parent generation which is selected through ranking strategy [35]. Mutation is used to generate new solutions (offspring). The method of mutation could vary dramatically with respect to specific problems, and it is a critical operation for EP, as it affects the behaviour of individuals greatly. The contemporary variant, Fast EP (FEP), uses a Cauchy mutation instead of Gaussian mutation. By introducing the Cauchy mutation, FEP is more likely to generate an offspring which is far away from its parent, due to its long flat tails. According to the experimental studies, the improvement enhances the global search ability of EP [1]. EP does not only conduct the mutation operation, but also capitalises on a continuous crossover operation in the evolution process.

  EP is comprehensively investigated for multi-objective problems, such as Fast Multi-Objective Evolutionary Programming [36], which uses fuzzy rank-sum concept and diversified selection, different multi-objective evolutionary programming [37], which has been used for detecting computer network attacks, and Multi-Objective Evolutionary Programming [38], which has been successfully applied in combined economic emission dispatch and economic emission dispatch problems in complex power systems.

- GP is a type of EAs which is on basis of the evolutionary progress developed by Koza[18]. It can solve various complex optimisations and searching problems successfully. GP is a domain-independent method that genetically breeds a population of computer programs to solve a problem. It is an extension of the GA, but the structures of the population in GP are not fixed-length character strings used in GA. In GP, the candidate solutions to the problem [39] are programs, which are expressed in genetic programming as syntax trees rather than as lines of code. Trees can be easily evaluated in a recursive manner. Every tree node has an operator function, and every terminal node has an operand, making mathematical expressions easy to evolve and evaluate. With tree structures representing trial solutions, GP can be applied to search for the optimal solution, and also used to search for mathematical functions to describe an unknown model.

  The genetic operators used in GP include crossover, mutation, reproduction, gene duplication, and gene deletion. Mutation affects the population individually. If one individual is selected to perform crossover, it will simply switch one of its nodes with another that is from another individual in the population. With the tree-based representation, it can only replace the node's information, or it can replace a whole branch from the selected node. In the latter case, the crossover is regarded as a replacement of the whole branch. These two operators are applied to the chromosome in each generation.

- ES was invented in 1963 by Ingo Rechenberg, Hans-Paul Schwefel at the Technical University of Berlin (TUB) while searching for the optimal shapes of bodies in a flow [19][40]. ES is a kind of EAs where individuals are encoded by a set of real-valued "object variables". For each object variable, an individual also has a "strategy variable" which determines the degree of mutation to be applied to the corresponding object variable. The strategy variables also mutate, allowing the rate of mutation of the object variables to vary. The simplest evolution strategy operates on a population size of two: the current point (parent) and the result of its mutation (child). In each generation, a vector of object variables is created by mutating the parent with an identical standard deviation. The fitness value of the child individual is compared with that of its parent, and the one with the better fitness value survives. This selection mechanism is identified as (1+1)-ES. However, this mechanism may result in converging to premature results during the optimisation process, due to a lack of diversity. To overcome this drawback, a multi-membered ES with μ parents, (μ+1)-ES, was proposed by Rechenberg [41]. In this method, μ parent individuals can participate in the generation of one offspring individual. This operation has a similar effect to the crossover process in GA. There are other ES variants based on this improvement, such as (1+λ)-ES, where λ mutants are generated from the same parent; $(\mu + \lambda)$-ES, where the best μ individuals are produced from the union of parents and offspring ($i.e.$ the selection operates on the joined set of parents and offspring); and (μ, λ)-ES, where only the best μ offspring individuals are used to form the next parent generation [42].

ES has been applied to solve practical problems in the field of engineering, such as the optimisation problem in power plant [43].

Swarm intelligence-base EAs

With the exception of the class of EAs which adopts the biological evolutionary progress, there are also some EAs which are based on swarm intelligence. Swarm intelligence is the collective behavior of decentralized self-organized systems [44]. These systems are typically made up of a population of simple agents. These agents follow very simple rules, and no centralized control structure dictates how individual agents should behave. Although there is no centralized control or the provision of a global model, agents interact locally with one another and with their environment. Hence, local interactions between such agents lead to the emergence of “intelligent” global behavior. These coherent global patterns could be unknown to the individual agents. A certain degree of randomness is applied to the local interactions between agents, in order to keep diversity among the swarm. Natural examples of swarm intelligence include bird flocking, ant colonies, animal herding, bacterial growth, and fish schooling, *etc*.

- Particle Swarm Optimisation (PSO) is a stochastic optimisation technique developed by Eberhart and Kennedy [45][44]. It is inspired by computer simulations of various interpretations of the movement of organisms in a flock of birds or a school of fishes. PSO is popular in the research field of optimisation in the last decade for its simplicity of implementation, few parameters and high convergence rate [46].

  In PSO, the population is called a *swarm*, and each individual of the population is called a *particle*. PSO works on the social behavior of particles in the swarm, by remembering the best location of itself and the best experience of other individuals in the swarm. The particles alter their velocities according to their records at each iteration. Assume that the search space is N-dimensional, and the number of particles is n_p. The position vector and velocity vector of the particle i (i=1,2,...,n_p) are denoted by $\mathbf{Z}_i$=(z_{i1},...,z_{iN}) and $\mathbf{V}_i$=(v_{i1}, ...,v_{iN}) respectively. The best position of particle i and the fittest particle found so far in the swarm are represented by $\mathbf{P}_{\mathrm{l}i}$=($p_{\mathrm{l}i1}$,...,$p_{\mathrm{l}iN}$) and $\mathbf{P}_\mathrm{g}$=($p_{\mathrm{g}1}$,...,$p_{\mathrm{g}N}$) respectively. At each iteration, every component of the velocity vector of particle i is updated according to:

  $$v_{ij}^{n+1} = wv_{ij}^{n} + c_{\mathrm{f1}}\zeta_1(p_{\mathrm{l}ij} - z_{ij}^{n}) + c_{\mathrm{f2}}\zeta_2(p_{gj} - z_{ij}^{n}) \tag{1.2.1}$$

  And each component of the position vector is updated according to the following equation:

  $$z_{ij}^{n+1} = z_{ij}^{n} + v_{ij}^{n+1} \tag{1.2.2}$$

  where n denotes the iteration number; w is the inertia weight [47]; ζ_1 and ζ_2 are random numbers in the range [0,1]. Constants c_{f1} and c_{f2}, called acceleration factors, are used to adjust particles' trajectory using its own previous best position and the best solution found in the group. The velocity obtained from (1.2.1) makes particles to some extent move towards the global optimum. The range of v_{ij} is denoted by [v_min, v_max], while that of z_{ij} is [z_min, z_max].

An important variant of standard PSO utilized a constriction factor, which was proposed by Clerc [48]. By reducing the inertia weight w in each iteration, it ensures the stability of convergence, and it leads to higher quality of the solutions compared with the standard PSO when solving unimodal functions, which have only one local minimum. However, an over-decreased inertia weight reduces the velocity of particle, *i.e.*, the particle is trapped more easily at local optima in multimodal optimisation problems which have a number of local optima. Other variants of the PSO have also been developed in recent years, such as Particle Swarm Optimisation with Passive Congregation [49], Unified Particle Swarm Optimisation, Fully Informed Particle Swarm and most recently, Cooperative Particle Swarm Optimisation (CPSO) [50] and Comprehensive Learning Particle Swarm Optimisation (CLPSO) [51]. These algorithms have produced encouraging results in both benchmark testing and real-world applications.

- Ant colony optimisation algorithm (ACO) is another approach which is based on swarm intelligence [52]. It models ant colony behaviour, including ant foraging behaviour, brood sorting, nest building, and self-assembling. Ants wander randomly, and deposit pheromone trails on the way back to the colony after finding food. If other ants find a pheromone path, they are more likely to discontinue their random travel, and follow the trail until reaching the food source. Subsequently, they reinforce the pheromone trails. The pheromone trails evaporate with time, resulting in a reduction of the strength of attraction. This process is simulated by ACO to find an optimal solution.

  ACO has been applied to many combinatorial optimisation problems, in which the set of feasible solutions is discrete or can be reduced to discrete. Taking Traveling Salesman Problem (a popular combinatorial problem) as an example, the solution obtained by ACO is much better than the one found by a GA [53]. After more than ten years of studies, both its application effectiveness and its theoretical background have made ACO a successful EA in the combinatorial optimisation domain [54]. Besides, the applications of ACO range from quadratic assignment to protein folding and vehicles routing. A number of derived methods have been adopted to solve dynamic problems, stochastic problems, multi-targets and parallel implementations [55].

- Recently, a Group Search Optimiser (GSO) was developed at the University of Liverpool [56], which was inspired by animal searching behaviour and group living theory [57]. The strategies of information-sharing [58] and producer-scrounger models [59] are employed in GSO. There are mainly three roles in a group: producer, scrounger and ranger. Producer, regarded as the leader of the group, is usually located in promising area, and seeks food source for the rest of the group. There is only one producer in the group. In each iteration, the producer is renewed and are selected from the best members. On the other hand, scroungers follow the producer and find opportunities to share the information found by producer and search resource uncovered by other scroungers. Apart from the producers, 80% of the rest members are randomly selected as scroungers, and the scroungers are renewed in each iteration as well. The remainder members are rangers. They walk randomly in the searching space, by dispersing themselves from their original positions. This behavior increases the diversity in the group, and discourages members from being trapped in local optima. Concepts from this framework are employed metaphorically to design optimum searching strategies for solving continuous optimisations. GSO is extended to solve multi-objective optimisation problems using multiple producers. Scroungers search around one of the producer and one of the non-dominated solutions. The number of the producers is equal to that of the objective functions. This method has been successfully applied to solve the problem of device placement in power systems [60].

1.3 Introduction to Learning Automata

The first learning automaton model was developed in mathematical psychology, and then developed by Bush $et\ al.$ [61], Atkinson $et\ al.$ [62] and Tsetlin [63]. Learning automata select their current actions based on past experiences learned through interaction with environment. In other words, learning automata should, by collecting and processing current information regarding the environment, be capable of changing their structure and parameters as time evolves, in order to achieve the desired goal or the optimal performance. The learning automata methods have already made a significant impact on many areas of system control and pattern classification, such as power system control [64][65], vehicle suspension control [66] and noise-tolerant learning of hyperplane classification [67], *etc.* They have also been investigated in solving optimisation problems. For instance, the method of continuous-action learning automata has been applied for stochastic optimisation [68][69]; the algorithm of genetic learning automata was proposed to resolve function optimisation problems [70][71]. However, few literature has reported that the learning automata methods, despite having a solid theoretical background, can outperform EAs in solving complex optimisation problems, particularly for high-dimensional optimisation problems. In addition, they have also been applied to solve multi-objective optimisation problems [72], but only limited in low-dimensional multi-objective problems.

1.3.1 Basic elements

A general block diagram of the interaction between environment and an automaton is provided in Fig. 1.1.

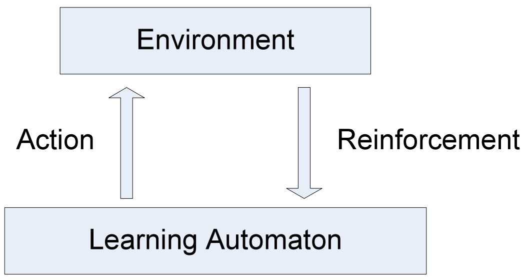


Figure 1.1: Block diagram of the interaction between a Learning Automaton and environment

The learning process of learning automata can be simply described as follows [73]: during the interaction with environment, the automaton randomly selects an action from the action set that includes the possible actions the automaton can take at its current state, based on a probability distribution. The probabilities of selecting the actions are the same initially. Then the environment generates a response to the action, named reinforcement signal. According to the response, the automaton updates the action probability distribution. The algorithm used to update the action probabilities is called the learning algorithm or reinforcement scheme. Afterwards, a new action is selected according to the updated probability distribution. Regarding this action, a response is elicited by the environment, and the procedure is repeated. It can be seen that the implementation of learning automata mainly concentrates on three aspects: how to select actions, learning algorithm and reinforcement signal, as given in Fig. 1.2. Unlike EAs that mainly apply the concept of biological evolutionary concept, learning automata methods are based on action selection and probability distribution.

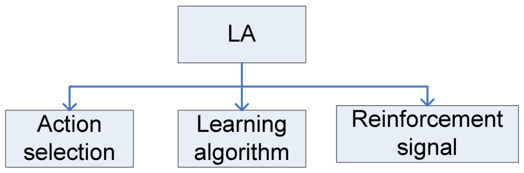


Figure 1.2: The three aspects of LA

The automaton

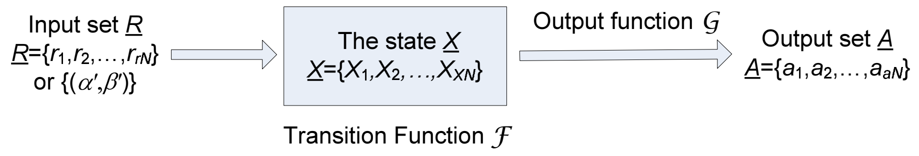

Input set $\underline{R}$
$\underline{R}$={r_1,r_2,…,r_{rN}}
or {(α',β')}
The state $\underline{X}$
$\underline{X}$={X_1,X_2,…,X_{XN}}
Output function $\mathcal{G}$
Output set $\underline{A}$
$\underline{A}$={a_1,a_2,…,a_{aN}}
Transition Function $\mathcal{F}$

Figure 1.3: The illustration of an automaton

A learning automaton is an adaptive decision-making unit situated in a random environment. The learning automaton learns the optimal action through repeated interactions with its environment. The illustration of an automaton is given in Fig. 1.3. Being an adaptive discrete machine, an automaton can be described as:

$$\{\underline{X}, \underline{A}, \underline{R}, \underline{P}, \mathcal{F}, \mathcal{G}, \mathcal{T}\} \tag{1.3.1}$$

These entities are described precisely as follows:

- The *state* of the automaton at instant n, denoted by $X(n)$, is an element of the finite set:

$$\underline{X} = \{X_1, X_2, \ldots, X_{XN}\} \tag{1.3.2}$$

- The *output* or *action* of an automaton at instant n, denoted by $a(n)$, is an element of the finite set:

$$\underline{A} = \{a_1, a_2, \ldots, a_{aN}\} \tag{1.3.3}$$

- The input to an automaton at instant n, denoted by $r(n)$, is an element of a set $\underline{R}$, which could be either a finite set or an infinite set, such as an interval on the real line. Thus,

$$\underline{R} = \{r_1, r_2, \ldots, r_{rN}\} \text{ or } \underline{R} = \{(\tilde{\alpha}, \tilde{\beta})\} \tag{1.3.4}$$

  where $\tilde{\alpha}$ and $\tilde{\beta}$ are real numbers.

- The probability of choosing action i at instant n, denoted by $p_i(n)$, is an element of the finite set:

$$\underline{P} = \{p_1, p_2, \ldots, p_{aN}\} \tag{1.3.5}$$

- $\mathcal{F}(\cdot, \cdot)$ is transition function, which determines the state at instant $(n + 1)$ regarding the state and input at instant n:

$$X(n+1) = \mathcal{F}[X(n), r(n)] \tag{1.3.6}$$

  or $\mathcal{F}$ is a mapping from $\underline{X} \times \underline{R} \to \underline{X}$. $\mathcal{F}$ could be either deterministic or stochastic.

- Output function $\mathcal{G}(\cdot)$ determines the output of the automaton at any instant n according to its current state:

$$a(n) = \mathcal{G}[X(n)] \tag{1.3.7}$$

  or $\mathcal{G}$ is a mapping $\underline{X} \to \underline{A}$. It is again either deterministic or stochastic.

- $\mathcal{T}$ represents the reinforcement scheme (learning algorithm), which determines the action probabilities at instant $n + 1$ regarding the probabilities at instant n:
$$p(n+1) = \mathcal{T}[p(n)] \tag{1.3.8}$$

Basically, the automaton takes in a sequence of inputs and outputs a sequence of actions with respect to the observation time n. The working of an automaton can be described as follows: given an initial state $X(0)$, action $a(0)$ is determined by function $\mathcal{G}$. Based on current state, the response generated by the environment, $r(0)$, and the transition function determine the next state $X(1)$. These operations are performed recursively, and in this way, the state sequence and action sequence are obtained according to any given input sequence [74].

In terms of functions $\mathcal{F}$ and $\mathcal{G}$, there are two types of automaton:

- Deterministic automaton. For this automaton, functions $\mathcal{F}$ and $\mathcal{G}$ are both deterministic mappings. It means that with a given initial state and a given input, the succeeding state and action are uniquely specified. The mappings of the two functions can be represented either in the form of matrices or graphs if the input set is finite. Corresponding to each input, the matrix or the graph can indicate how the present state transfers to a new state. In this case, the transition matrices consist of elements that are only either 0 or 1.

- Stochastic automaton. For this automaton, at least one of the functions $\mathcal{F}$ and $\mathcal{G}$ is stochastic. In other words, given an initial state and an input, there is no certainty concerning the succeeding state and action, which are associated with probabilities. If transition function $\mathcal{F}$ is stochastic, given the present state and input, the next state is at random, and $\mathcal{F}$ gives the conditional probabilities (or called transition probabilities) of reaching the various states. If the output function $\mathcal{G}$ is stochastic, given the present state, the action taken is at random and is determined by the conditional probabilities provided by $\mathcal{G}$.

  If the conditional probabilities provided by $\mathcal{F}$ and $\mathcal{G}$ are constant, $i.e.$ they are independent of n and the input sequence, the stochastic automaton is referred to as a fixed-structure stochastic automaton. In this case, the elements of the transition matrix are constants which take values in the interval [0,1], and each transition matrix is a stochastic matrix. On the other hand, if the transition probability at instant n is updated on the basis of the input at that instant, the automaton is called a variable-structure stochastic automaton. The elements of the transition matrix are in [0,1], but they are no longer constants any more, as they are updated with n.

The environment

Environment refers to the aggregate of all the external conditions and influences which would affect the automata [74]. The role of the environment is to establish the relationship between the actions taken by the automaton and the signal inputs to the automaton [69]. An environment can be mathematically defined as a triple $\{\underline{A}, \underline{C}, \underline{R}\}$. $\underline{A}$ represents a finite input set. $\underline{R}$ represents the output set, $i.e$ the environment's response to the action taken by the automaton. The environment is referred to as a P-model environment when its response belongs to the binary set $\{0, 1\}$; the environment is named as S-model environment when its response takes an arbitrary value in the closed segment [0,1]; Q-model environment refers to the environment whose response is one of a finite number of values in the interval [0,1]. The environment is characterized by $\underline{C}$, which represents the penalty probability, $i.e.$ the likelihood that the application of an action to the environment will result in a penalty output. If the penalty probabilities are constant, the environment is said to be stationary; otherwise, it is nonstationary.

Reinforcement schemes

Reinforcement schemes are the methods used to update action probabilities at each instant. They were originally proposed to model animal learning [61], but later it was found that they can be applied in the field of learning automata successfully. Choosing reinforcement scheme is a crucial factor that affects the performance of learning automata [75]. In general, a reinforcement scheme can be represented by:

$$\underline{p}(n+1) = \mathcal{T}[\underline{p}(n), r(n), a(n)] \tag{1.3.9}$$

where $\mathcal{T}$ is a learning operator; $r(n)$ and $a(n)$ represent the input to the automaton and the action taken by the automaton at instant n, respectively.

The basic idea behind a reinforcement scheme can be described as follows: if the automaton selects an action a_i at instant n and receives a nonpenalty input, the action probability of the action a_i, $i.e.$ $p_i(n)$, will be increased, and the probabilities for all other actions will be decreased. In this case, the change occurring on $p_i(n)$ is called reward. For a penalty input, $p_i(n)$ is decreased and other action probabilities are increased. At this moment, the change applied on $p_i(n)$ is called penalty.

If $p_i(n+1)$ is a linear function of $p_i(n)$, the reinforcement scheme can be described as linear, such as Linear Reward-Inaction (L_{R-I}) algorithm [74], which is known to be very effective in many applications. The L_{R-I} updates the action probabilities as follows (assuming $a(n) = a_i$):

$$\begin{aligned} p_i(n+1) &= p_i(n) + \tilde{\lambda} r(n)(1 - p_i(n)) \\ p_j(n+1) &= p_j(n) - \tilde{\lambda} r(n) p_j(n), \text{ for all } j \neq i \end{aligned} \tag{1.3.10}$$

On the other hand, if $p_i(n+1)$ is a nonlinear function of $p_i(n)$, the reinforcement scheme is said to be nonlinear. Several non-linear schemes have been suggested by Viswanathan and Narendra [76], Lakshmivarahan and Thathachar [77]. Sometimes, it is advantageous to use different schemes to update $p_i(n)$, while the selection of the reinforcement scheme depends on the intervals the value of $p_i(n)$ lies in.

1.3.2 Several learning automata methods

Although each individual of the learning automata is an independent unit in a simple structure, a group of learning automata can be connected in a way which would be suitable for tackling complex learning problems. Systems built with LA have been successfully employed in many difficult learning situations over the years. This has also led to the concept of LA being generalized in a number of directions, in order to handle various learning problems [78].

Finite Action-set Learning Automaton (FALA)

The original notion of learning automaton is derived from what is called finite action-set learning automaton (FALA) [74]. FALA is the learning automaton for which the number of possible actions is finite, $i.e.$ the size of the action-set is finite. This type of learning automaton has been studied extensively.

A learning automaton is an adaptive decision-making device that learns the optimal action out of a set of actions through repeated interactions with a random environment [78], as described in Section 1.3.1. There are two characteristics of the learning automata: 1) the action choice is based on a probability distribution over the action-set; 2) and the probability distribution is updated at each instant based on the reinforcement feedback from the environment.

The operation of FALA consists of repetitions of the following several steps: choosing action randomly, receiving reinforcement from environment and updating action probability. The process can be specifically described as follows: at each instant n, the automaton chooses an action a randomly, based on its current action probability distribution, $\underline{p}(n) = (p_1(n), \ldots, p_{aN}(n))^T$, where $p_i(n) = \text{prob}[a(n) = a_i]$ and $\sum_{i=1}^{aN} p_i(n) = 1, \forall n$. The action chosen by the automaton is the input to the environment. The environment responds to the action with a stochastic reaction or reinforcement, $r(n)$, which is the input to the automaton. The higher the value of the reinforcement signal, the more desirable the action. Let

$$F_i = E[r(n)|a(n) = a_i], i = 1, \ldots, aN \tag{1.3.11}$$

F_i is often referred to as reward probability of action a_i. Define an index I by $F_I = \max\{F_i\}$. Then the action a_I is called the optimal action. The learning automaton has no knowledge of the reward probabilities. The automaton's objective is to identify the optimal action. The goal is achieved through a learning algorithm that updates, at each instant n, the action probability distribution $\underline{p}$, through the most recent interaction with the environment, namely, using the pair $\{a(n), r(n)\}$.

Continuous Action Learning Automaton (CALA)

In FALA, the action-set is always considered to be finite. However, in many applications, one needs to learn a real-valued parameter. For instance, if the problem is to learn the 'optimal' parameter which maximises a performance index, the actions of the automaton could be the possible values of the parameter. In order to use FALA to resolve this learning task, the process of discretization is required. However, the discretization process can result in inaccurate results or a large computation load if finer discretization is applied. Therefore, a continuous space of actions is preferred in this case. One such model is called continuous action learning automaton (CALA) [68].

The action-set of CALA is the real line. A real number a is chosen at instant n based on present action probability distribution, which is a normal distribution with mean $\mu(n)$ and standard deviation $\sigma(n)$. Both $\mu(n)$ and $\sigma(n)$ are updated at each instant based on the reinforcement received from the environment. The objective of CALA is to learn the value of a which maximises the function $F(a) = E[r_a \mid a]$, which is considered as the measurement of a. The action probability distribution, denoted as $N(\mu(n), \sigma(n))$, is supposed to be able to converge to $N(a^*, 0)$, where a^* is the optimal solution for maximising F. However, in order to avoid the algorithm sticking at a non-optimal point, $\sigma(n)$ should converge to σ_L instead of 0.

The operation of CALA can be summarised as follows: at each instant n, a real number $a(n)$ is chosen at random based on its present action probability distribution $N(\mu(n), \phi(\sigma(n)))$, where $\phi(\sigma)$ is defined as:

$$\phi(\sigma) = \begin{cases} \sigma_L & \text{for } \sigma \leq \sigma_L \\ \sigma & \text{for } \sigma > \sigma_L > 0 \end{cases} \tag{1.3.12}$$

Then, the automaton receives the reinforcement signals from the environment with respect to actions $\mu(n)$ and $a(n)$. Assume the two reinforcement signals are denoted as r_μ and r_a respectively. Then, μ and σ of the probability distribution are updated as follows:

$$\mu(n+1) = \mu(n) + \tilde{\lambda}\frac{(r_a - r_\mu)}{\phi(\sigma(n))}\frac{(a(n) - \mu(n))}{\phi(\sigma(n))} \tag{1.3.13}$$

$$\sigma(n+1) = \sigma(n) + \tilde{\lambda}\left[\left(\frac{r_a - r_\mu}{\phi(\sigma(n))}\right)^2 - 1\right] + \tilde{\lambda}\{C_{\mathrm{p}}[\sigma_L - \sigma(n)]\} \tag{1.3.14}$$

where $\tilde{\lambda}$ $(0 < \tilde{\lambda} < 1)$ is the step size parameter for learning; and C_{P} is a large positive constant. The CALA algorithm can be used for optimisation problems [79], and the randomness in choosing the next parameter value makes the algorithm explore better search directions.

Generalized Learning Automaton (GLA)

GLA is an extended learning automaton which accepts an extra input. In GLA, the inputs to the automata are not only reinforcement signals from the environment, but also a context vector, which is the state of the environment that is made available to the learning system as an additional input. The goal of GLA is to learn an optimal mapping which associates different context vectors with action selection. To be specific, the action is selected in response to context vector input that is provided by the environment, through interacting with the environment in the form of selecting actions and receiving reinforcement signals. A single GLA can be described by the tuple $< \underline{X}, \underline{A}, \underline{R}, \underline{S}, g, \mathcal{T} >$. $\underline{X}$ is the set of all context vectors that can be input to the GLA; $\underline{A}$ is the (finite) set of outputs or actions of GLA; $\underline{R}$ is the set of values that the reinforcement signal can take (usually in the interval [0,1]); g is the probability generating function; and $\underline{S}$ is the internal state which is a vector of real numbers. $\mathcal{T}$ is the learning algorithm that updates $\underline{S}$. The individual context vector, which is the member of $\underline{X}$, is denoted by X.

The action probabilities of GLA are generated by:

$$\mathrm{Prob}[a(n) = a_i \mid \underline{S}, X] = g(X, a_i, \underline{S}) \tag{1.3.15}$$

where $a_i \in \underline{A}$, function g satisfies $g(X, a_i, \underline{S}) \geq 0, \forall, a_i, \underline{S}, X$, and $\sum_{i=1}^{m} g(X, a_i, \underline{S}) = 1, \forall, \underline{S}, X$. At each instant n, the learning algorithm $\mathcal{T}$ updates $\underline{S}(n)$ based on $X(n)$, $\underline{S}(n)$, $r(n)$, and the action chosen by GLA, $a(n)$. It can be seen that the updating process is dependent on the context vector $X(n)$, which is the main characteristic of GLA.

The motivation of defining GLA is to make it be able to tackle associative reinforcement learning problems directly. With the same state vector $\underline{S}$, the probabilities of choosing different actions can (and most often, would) be dependent on the context vector. This explains why the probability generating function of GLA is also dependent on the context vector X, as well as the state $\underline{S}$. In GLA, state $\underline{S}$ and function g are together determining a representation for the mapping of context vectors to action probability distributions.

An example of the probability generating function is given as follows: suppose that context vector X belongs to $\Re^N$, if there are aN actions, then the internal state can be denoted as a set of aN vectors, $\underline{S} = (\bar{S}_1^T \dots \bar{S}_{aN}^T)^T$ where $\bar{S}_i \in \Re^N$ and $\bar{S}_i = (\bar{s}_{i1}, \dots, \bar{s}_{iN(i)})^T$. The probability generating function can be described as follows:

$$g(X, a_i, \underline{S}) = \frac{\exp(-X^T \bar{S}_i)}{\sum_j \exp(-X^T \bar{S}_j)} \tag{1.3.16}$$

In general, GLA is to learn the desired mapping between context vectors and actions. In other words, the objective of learning is to map an action to $\underline{S}$, so that the expectation of reinforcement is the maximum. The goal can be defined as $F(\underline{S}) = \max E[r \mid \underline{S}]$. The learning algorithm $\mathcal{T}$ can update the state $\underline{S}$ according to the following equation [80][78]:

$$\begin{aligned}&\bar{S}_i(n+1) = \bar{S}_i(n) + \tilde{\lambda} r(n) \tfrac{\partial \ln g(X(n), a(n), \underline{h}(\underline{S}(n)))}{\partial \bar{S}_i} \\ &+ \tilde{\lambda} K_i [h_i(\bar{S}_i(n)) - \bar{S}_i(n)]\end{aligned} \tag{1.3.17}$$

where

$$\underline{h}(\underline{S}) = [h_1(\bar{S}_1), \dots, h_{aN}(\bar{S}_{aN})]^T$$

with

$$h_i(\eta) = \begin{cases} L_i & \text{for } \eta \geq L_i \\ \eta & \text{for } |\eta| \leq L_i \\ -L_i & \text{for } \eta \leq -L_i \end{cases} \tag{1.3.18}$$

where $\tilde{\lambda} > 0$ is a learning parameter; and $L_i, K_i > 0$ are constants.

Parameterized Learning Automaton (PLA)

One feature of GLA is the use of the internal state of the automaton, which is a vector of real numbers updated at each instant. The probabilities of selecting different actions are computed from the internal state, together with a given context vector, using a probability generating function. This feature can be introduced to FALA, even in the absence of context vectors. Then, the action probabilities, which are updated by a learning algorithm, can be regarded as being parameterized in terms of real numbers (i.. internal state). Such a FALA is referred to as Parameterized Learning Automaton (PLA) [81].

PLA has an internal state vector $\underline{S}$ which is composed of real numbers. Based on the value of $\underline{S}$, the probabilities of selecting various actions are calculated using a probability generating function g. For example, the probabilities of selecting the i^{th} action could be computed as:

$$p_i = g(\bar{S}, i) = \frac{\exp(\bar{S}_i)}{\sum_{j=1}^{aN} \exp(\bar{S}_j)} \tag{1.3.19}$$

where $\bar{S}_i$ is the i^{th} component of $\underline{S}$. It may be noticed that in GLA, both the internal state and the context vector determine the action probability. However, in the case of PLA, the internal state is the only parameterization of the action probability generation function.

The goal of the learning algorithm is to maximise

$$F(\underline{S}) = E[r|\underline{S}] \tag{1.3.20}$$

subject to $|\bar{S}_i| \leq L_i$, $i = 1, \ldots, aN$, where $L_i > 0$ is a constant.

To ensure convergence to the global maximum, a learning algorithm, which is used to update the internal state, can be described below:

$$\bar{S}_i(n+1) = \bar{S}_i(n) + \tilde{\lambda} r(n) \frac{\partial \ln g(\bar{S}_i(n), a(n))}{\partial \bar{S}_i(n)} + \tilde{\lambda} h'(\bar{S}_i(n)) \tag{1.3.21}$$

where $h'(\cdot)$ is the derivative of $h(\cdot)$, defined as:

$$h(\eta_i) = \begin{cases} -K(\eta_i - L_i)^{2J} & \text{for} \quad \eta_i \geq L_i \\ 0 & \text{for} \quad |\eta_i| \leq L_i \\ -K(\eta_i - L_i)^{2J} & \text{for} \quad \eta_i \leq -L_i \end{cases} \tag{1.3.22}$$

where K is a positive constant and J is a positive integer.

1.4 Overview of this Thesis

This thesis is structured as follows:

Chapter 2 introduces a novel optimisation algorithm, Function Optimisation by Learning Automata (FOLA). It mainly includes the following aspects employed in FOLA: dimensional search, path value, action selection, reinforcement signal, cell value and state memory. FOLA has been compared with a number of classical Evolutionary Algorithms (EAs) on 13 high-dimensional benchmark functions, which represent a wide range of challenging optimisation problems. The simulation results presented in this chapter have shown that FOLA is able to undertake search in continuous states and achieve accurate solutions efficiently. The analysis of FOLA, in terms of convergence characteristics, computation time and parameters, is also carried out in the chapter. To further evaluate FOLA's performance, it is also compared with two popularly used EAs and four newly-proposed EAs, on 9 complex multimodel benchmark functions. The experimental results demonstrate the superiority of FOLA over other algorithms for most benchmark functions, in both the convergence rate and the accuracy of finding optimal solutions. FOLA is able to reduce computation time greatly, especially for high-dimensional functions.

Chapter 3 presents a multi-objective optimisation by learning automata (MOLA). The approaches employed in MOLA, including reinforcement scheme, forming of Pareto set, and the process of searching and learning, are introduced in detail in this chapter. MOLA has been compared with two popular weighted-sum based multi-objective EAs on four multi-objective benchmark functions that comprise low and high-dimensional models, convex and non-convex models, and continuous and discontinuous models respectively. The simulation results of these algorithms have been analysed with respect to the accuracy of the experimental results and the range of the Pareto front. MOLA is also compared with multi-objective evolutionary algorithm based on decomposition (MOEA/D) and non-dominated sorting genetic algorithm II (NSGA-II), on thirteen multi-objective benchmark functions, which are difficult to resolve due to complicated Pareto set. The simulation results have presented the great superiority of MOLA over MOEA/D and NSGA-II, as MOLA can obtain more accurate Pareto optimal solutions, and find wider range of the Pareto front.

Chapter 4 describes two applications of FOLA. The first concerns with the optimal power system dispatch and voltage stability problem. The problem is to reduce the fuel cost whilst enhancing the voltage stability of the power system. Simulation studies are undertaken on the standard IEEE 30-bus and 57-bus power systems respectively. The advantages of FOLA have been demonstrated by comparing its performance with that of improved PSO and GA. The second case concerns the renewable energy of wind power. Wind power penetrated power systems bring new challenges to power system operation since the dynamic nature of wind power. In the thesis, FOLA is applied to tackle the optimal power flow problem which aims to achieve economic power dispatch and voltage stability enhancement in dynamic wind power penetrated systems. FOLA is compared with the improved PSO and GA, on the modified IEEE 30-bus and 57-bus power systems respectively, which are penetrated with time-varying wind power. The experimental results have demonstrated that FOLA outperforms PSO and GA, as it tracks the changing system configuration more rapidly and accurately than the improved PSO and GA. In addition, FOLA is compared with CLPSO and CPSO based on the modified IEEE 118-bus power system. FOLA is able to minimise the fuel cost and enhances the voltage stability of the power system more efficiently in comparison with the other two algorithms.

Chapter 5 presents three applications of MOLA. The first two are concerned with the increasing attention on pollutant emission and the use of wind power. The chapter applies MOLA to solve the problem of economic emission dispatch and voltage stability enhancement in wind power penetrated power systems, which are incorporated with fixed-speed and variable-speed wind generators respectively. MOLA is compared with MOEA/D and NSGA-II on the modified IEEE 30-bus power system and new England test power system respectively. The simulation results have demonstrated the superiority of MOLA over NAGA-II and MOEA/D. The third application of this chapter concerns the deregulated market. The deregulation of the power market creates more competition and more trading mechanisms for market players. MOLA is applied to maximise social benefit and enhance voltage stability simultaneously in a deregulated electricity market. MOLA has been compared with MOEA/D and NSGA-II, based on the challenging optimisation problems on the IEEE 30-bus power system. In this application, the simulation results have again demonstrated the superiority of MOLA over MOEA/D and NSGA-II, as MOLA can find wider and evenly distributed Pareto fronts, and obtain more accurate non-dominated solutions.

Chapter 6 concludes this thesis based on the experimental results obtained in this study. The merits of FOLA and MOLA are further discussed, as well as their application capabilities. Additionally, this chapter includes suggestions for future work.

Appendix gives the details of single objective and multi-objective benchmark functions employed in the thesis, as well as the notations used in the thesis.

1.5 Contributions of the Research

Several contributions and outcomes made in this research are highlighted in this section.

- A novel optimisation algorithm, FOLA, is developed in this research, by improving the two main aspects of Fig. 1.2: action selection and learning algorithm. FOLA consists of multiple automata, where each automaton undertakes dimensional search on a selected dimension of the solution domain. FOLA has the ability of memorising history by estimating and updating the values of states that have been visited. With these approaches, FOLA is able to undertake search in continuous states and achieve accurate solutions efficiently. In contrast to EAs which adopt population-based search, FOLA reduces the computation complexity by sequential dimensional search. Unlike EAs which mainly memorise the global optimal solution and use it to guide search, FOLA memorises the previous performance evaluations, which can provide more information for future seach and thus increase the efficiency. The experimental results have demonstrated that FOLA has better performance than a number of EAs.

- Multi-objective optimisation by learning automata (MOLA) is presented to solve complex multi-objective optimisation problems. MOLA capitalises on the merits of the structure of multiple automata, the dimensional search, the dividing of the dimensional search domain into cells, state memories via which the search action is carried out, and the process of learning from the best solution and neighboring solutions. MOLA is able to find accurate non-dominated solutions and evenly distributed Pareto fronts, when solving the complex optimisation problems which have complicated Pareto set. The merits of MOLA have been demonstrated, in comparison with two latest development of multi-objective EAs, $i.e.$ MOEA/D and NSGA-II.

- The proposed FOLA has been applied to solve the power system dispatch and voltage stability problem. This problem has received considerable attention, and is widely used in power system operation and planning, due to that power system dispatch is an important factor from the perspective of cost in modern energy systems, and the voltage instability is a major power system weakness resulting in severe detriments with economical, technical and social dimensions. This problem can be formulated as a highly constrained complex optimisation problem, which has the nature of non-differential, non-linearity and non-convex. For this problem, the applicability of FOLA has been investigated based on standard IEEE 30-bus and 57-bus power systems. Comparison has been carried out among FOLA, improved GA and PSO. The experimental results show that FOLA has superior performance over the other two algorithms.

- With the increasing demand of power nowadays and the stress of the resources which can be used to generate power, renewable energy becomes more and more important in many countries, especially wind energy. Wind power interconnected to power systems brings new challenges to power system economic operation. It is imperative to study how to efficiently solve optimal power flow formulation which is integrated with wind power, so as to achieve an optimal solution to the specific power system objective functions. The application proposed here aims to achieve economic power system dispatch and voltage stability enhancement in dynamic wind power integrated systems. IEEE 30-bus, 57-bus and 118-bus power systems, which are integrated with time-varying wind power, have been employed to evaluate the performance of FOLA for the problem. The experimental results show that FOLA is able to optimise the problem efficiently.

- MOLA has been applied to solve the problem of economic emission dispatch and voltage stability enhancement in wind power integrated power systems, due to the increasing importance of pollutant emission and wind power. The IEEE 30-bus power system and new England test power system are modified to integrate fixed-speed and variable-speed wind power generators respectively. MOLA has been applied to solve the problem, and the simulation results have presented the superiority of MOLA over NAGA-II and MOEA/D.

- Deregulating the power market creates more competition and more trading mechanisms for market players. However, the emergence of deregulated electricity markets poses new challenges to the solution of the optimal power flow problem. MOLA is applied to maximise social benefit and enhance voltage stability in deregulated electricity market simultaneously. MOLA is compared with MOEA/D and NSGA-II, on the challenging optimisation problems on the IEEE 30-bus power system. MOLA can obtain accurate Pareto optimal solutions and find wider and evenly distributed Pareto fronts.

The publications produced from this research work are listed in this section as follows:

1. Q.H. Wu and H.L. Liao. Function Optimisation by Reinforcement Learning For Power System Dispatch and Voltage Stability. In *Proc. of PES General Meeting*, pages 1-8, Minneapolis, Minnesota, USA, 2010.

2. H.L. Liao and Q.H. Wu, Multi-Objective Optimisation by Reinforcement Learning. In *Proc. of IEEE World Congress on Computational Intelligence*, pages 3374-3381, Barcelona, Spain, 2010.

3. Q.H. Wu and H.L. Liao. High-dimensional Optimisation by Reinforcement Learning. In *Proc. of IEEE World Congress on Computational Intelligence*, pages 2808-2815, Barcelona, Spain, 2010.

4. H.L. Liao, Q.H. Wu, and L. Jiang, Multi-Objective Optimisation by Reinforcement Learning for Power System Dispatch and Voltage Stability. In *Proc. of PES Conference on Innovative Smart Grid Technologies Europe*, pages 1-8, Gothenburg, Sweden, 2010.

5. H.L. Liao and Q.H. Wu. Optimal Power Flow in Wind Power Integrated Systems using Function Optimisation by Learning Automata. In *Proc. of PES General Meeting*, pages 1-8, Detroit, MI USA, 2011.

6. M.S. Li, Q.H. Wu, H.L. Liao, W.J. Tang and Y.S. Xue. Optimal Power Flow with Environmental Constraints Using Paired Bacterial Optimiser. In *Proc. of PES General Meeting*, pages 1-8, Detroit, MI USA, 2011.

7. Z. Ji, J.R. Zhou, H.L. Liao and Q.H. Wu, A Novel Intelligent Single Particle Optimiser, *Chinese Journal of Computers*, 33(3):556-561, 2010.

8. Z. Ji, H.L. Liao, and Q H. Wu, *Particle Swarm Optimisation and Its Applications*, Science Press (China), 2009.

Part 1

Developments of Learning Automata-based Optimisation Algorithms

Chapter 2

Functional Optimisation by Learning Automata

2.1 Introduction

Due to the increasingly complex real-world optimisation problems and the insufficiency of classical optimisation algorithms in solving multi-modal problems, biologically inspired optimisation algorithms, which incorporate the behaviours of biological principle into their algorithmic framework [5], have been comprehensively investigated over the last few decades. Notably EAs, which are introduced in Section 1.2.2, have been widely applied for problem solving over the last twenty years, ranging from scientific problems to engineering applications. However, the performance of EAs is interfered by the randomness and the large population sized applied, which cause unexpected redundant computational load, and thus reduce the efficiency of the algorithms in many applications [32]. Besides EAs, learning automata methods have also been applied to resolve optimisation problems. For instance, the method of continuous-action learning automata has been applied for stochastic optimisation [68][69]; the genetic learning automata was proposed to solve function optimisation problems [70]. In contrast to EAs, the learning automata methods are less popularly applied for solving complex function optimisation problems, despite having a solid theoretical background and having a significant impact on many areas of systems control and pattern classification [64][66][67].

This chapter presents a novel algorithm: FOLA (Function Optimisation by Learning Automata). FOLA consists of multiple automata, in which the number of automata used is equal to the total number of dimensions of the solution domain. Each automaton is responsible for searching on one specified dimension, and dimensional states on this dimension are considered as the states of this automaton. Before an action is taken, the automaton, at a dimensional state, selects a path from two possible paths, according to the probability which is calculated from the path value that is estimated to indicate the potential of finding a better solution if the automaton searches down on this path. Then an action takes place by moving on the selected path with a certain step length. To evaluate the effectiveness of the action, a reinforcement signal, in a scale value, is generated by the environment.

In order to avoid a large amount of computation caused by the huge number of states involved in the continuous search domain, each dimension is divided into a certain number of cells, and each cell is assigned with a cell value. The cell value is updated based on its past values, and the path values and reinforcement signal obtained when the dimensional states located in the cell are visited. FOLA has been found capable and efficient in finding accurate solutions to complex optimisation problems. Its merits have been demonstrated, in comparison with other EAs, by evaluating it on a number of uni-modal or multi-modal benchmark functions respectively. The experimental results have shown that FOLA outperforms the other EAs, in terms of accuracy and efficiency, especially for solving high-dimensional optimisation problems.

2.2 The FOLA Method

The structure of the learning automata used in FOLA is illustrated in Figure 2.1. FOLA consists of N automata, where N is equal to the number of dimensions of the solution domain concerned in an optimisation problem. Each automaton is considered as an independent entity situated in an environment, and it is responsible for searching on a specified dimension. The ith learning automaton can be defined as $\langle \chi_i, A_i, r, P_i, \mathcal{T} \rangle$, in which χ_i, $\chi_i = \{x_i\}$, is the set of possible states on the ith dimension, and x_i is the dimensional state on the ith dimension, $x_i \in [x_{\min,i}, x_{\max,i}]$, where $x_{\min,i}$ and $x_{\max,i}$ represent the minimum and maximum values of the ith dimension respectively; A_i denotes the set of possible actions which the learning automaton can take on dimension i, and in FOLA, $A_i = \{a_{l,\eta}\}$, and $a_{l,\eta}$ indicates a search action taken to allow x_i to move on left path ($l = 1$) or right path ($l = 2$) by a step length η; r is a reinforcement signal in scalar value, generated by the environment to indicate the quality of the action of moving x_i in a step length on the selected path; P_i includes two probabilities, p_1 and p_2, where p_1 denotes the probability of selecting the left path or the right path on the ith dimension, and probability p_2 determines the step length of moving x_i towards the selected path; and $\mathcal{T}$ is a scheme adopted to calculate the probabilities of actions, P. The detailed discussion on the action selection, reinforcement signal and scheme $\mathcal{T}$ will be presented in Section 2.2.1.

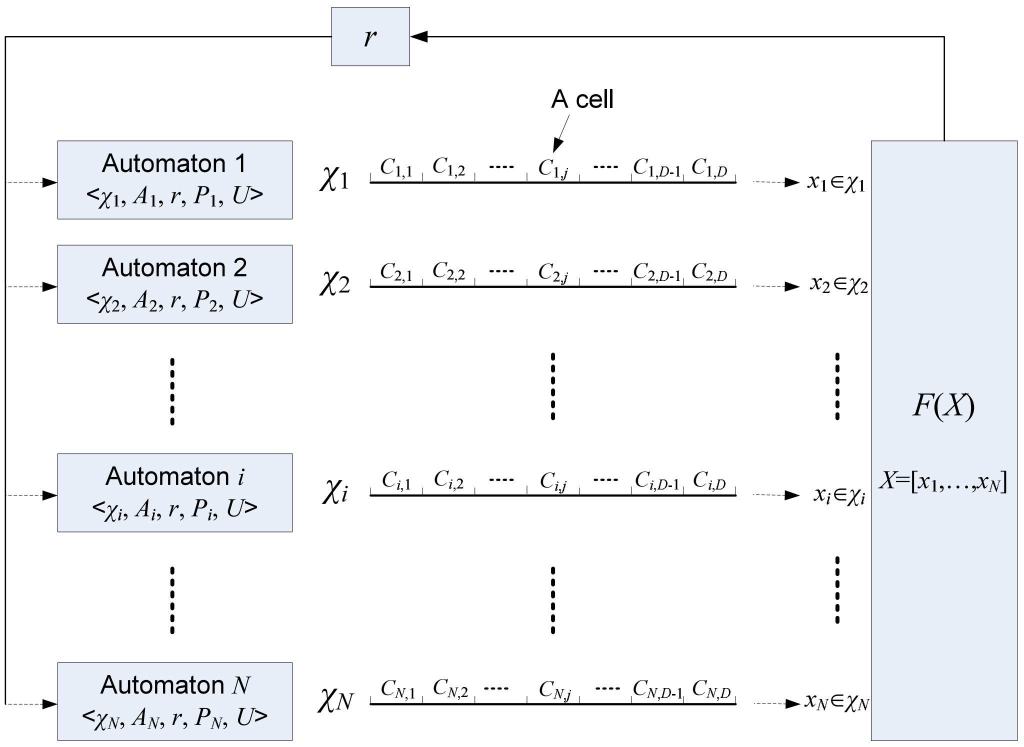


Figure 2.1: The structure of learning automata for FOLA

$F(X)$ shown in Figure 2.1 is an N-dimensional function minimisation (or maximisation) problem to be resolved subject to constraints applied to $F(\cdot)$ or/and X, where $F(\cdot)$ is a continuous objective function used by the environment to generate reinforcement signal r, and X is a solution in the N-dimensional space, denoted by $X = [x_1, \cdots, x_i, \cdots, x_N]$, where x_i is the dimensional state of dimension i, that is, $x_i \in \chi_i$. Note that X is the combination of N dimensional states and denotes the state of all automata, thus it is also called a state.

Since the dimensional states are continuous variables and the search space contains an infinite number of possible solutions, it is difficult to define state values. In order to resolve the problem of the large number of states that causes a huge amount of computation, each dimension is divided into D cells, in other words, χ_i is divided into D subsets, and each subset consists of all the dimensional states located in the cell. For an N-dimensional search space, there are $N \times D$ cells. Each cell is denoted as $c_{i,j}$, where $i \in \{1, 2, \cdots, N\}$, $j \in \{1, 2, \cdots, D\}$. The width of a cell in the ith dimension is denoted by $w_{\mathrm{c},i}$ and $w_{\mathrm{c},i} = [x_{\max,i} - x_{\min,i}]/D$. Each cell $c_{i,j}$ has its cell value, denoted as $V(x_i)|_{x_i \in c_{i,j}}$, which is updated by the path values and reinforcement signal when visiting the dimensional states located in the jth cell. The cell values will be explained in detail in Section 2.2.1.

2.2.1 An automaton and its reinforcement scheme

Path value

Before a search action is taken, a possible path is selected to estimate the potential of finding a better solution if the automaton searches down on the path from its dimensional state. Two estimated path values, denoted by $L_l(x_i)$, $l = 1, 2$, are to be found with respect to dimensional state x_i, for selecting one of two possible directions: moving on the left path or on the right path. The path values are determined by the cell values on the path. As shown in Fig. 2.2, the estimate of $L_1(x_i)$ is determined by the values of k adjacent cells on the left path, $V(c_{i,j-1})|_{x_i \in c_{i,j-1}}, V(c_{i,j-2})|_{x_i \in c_{i,j-2}}, \cdots, V(c_{i,j-k})|_{x_i \in c_{i,j-k}}$, where k is a pre-set integer; $c_{i,j}$ is the cell that dimensional state x_i locates in; j can be formulated as

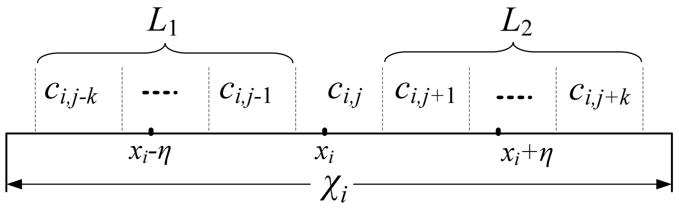


Figure 2.2: The two possible paths taken by a search starting at dimensional state x_i on the ith dimension

$j = \text{floor}((x_i - x_{\min,i})/w_{c,i})$ and $\text{floor}(y)$ denotes the greatest integer less than real value y. The value of a path can be estimated as follows:

$$L_l(x_i) = (1-\lambda_1)\sum_{m=1}^{k-1}\lambda_1^{m-1}v^*_{l,m} + \lambda_1^{k-1}v^*_{l,k} \qquad l = 1,2 \tag{2.2.1}$$

where $v^*_{l,m}$ denotes the mth element of the vector in which the k cell values, locating on path l, are reordered in descending order, and in other words, $v^*_{l,m}$ represents the mth largest value among the k cell values; λ_1 denotes a weight introduced for the values of the cells on the lth path, $0 \le \lambda_1 \le 1$ and $(1-\lambda_1)\sum_{m=1}^{k-1}\lambda_1^{m-1} + \lambda_1^{k-1} = 1$, subject to $(1-\lambda_1)\lambda_1^{k-2} \ge \lambda_1^{k-1}$, as given in [82]. It can be seen that the larger the cell value is, the greater it influences on the path value, which is the reference for later path selection, as discussed in Section 2.2.1. In other words, if the cells locating on the path have larger cell values, the path will be assigned with a larger path value, which suggests that the path is more likely to be selected for subsequent search.

Action selection

Action selection employs two probabilities, p_1 and p_2. The former is used to select a path which the current dimensional state moves to, while the latter is applied to select the cell on the selected path. With the availability of the path values, the left path L_1 or the right path L_2 is chosen with a probability given as follows:

$$p_1(L_l(x_i)) = \frac{e^{L_l(x_i)/\tau}}{\sum_{s=1}^{2} e^{L_s(x_i)/\tau}} \qquad l = 1,2 \tag{2.2.2}$$

This probability is similar to that used for a parameterised learning automaton reported in [73], apart from using τ which denotes a temperature. Temperature τ,

$\tau > 0$, is a parameter chosen to balance the exploration and exploitation [83]. A high temperature causes all paths to be equiprobable, while a low temperature makes a greater difference in selection probability for paths that differ in path values. As $\tau \to 0$, the selection probability of the path, which has a higher path value, approaches one, while that of the path which has a smaller value is approximately zero. In this case, the selection strategy becomes more selective, only choosing a path with a higher path value. Setting τ requires the knowledge of possible path values. For most of the dimensional states in FOLA, the reinforcement signal is either 0 or 1, as discussed later in Section 2.2.1, thus it would be better to set $\tau \leq 1$ to make a greater difference in selection probability for the paths which have different path values.

Suppose the right path is selected. Then a cell, which the current dimensional state moves to, is selected, among the k cells located on the path, according to the probability that is calculated from the cell values of the k cells as follows:

$$p_2(V(c_{i,j+s})|_{x_i \in c_{i,j+s}}) = \frac{e^{V(c_{i,j+s})|_{x_i \in c_{i,j+s}}/(2\tau)}}{\sum_{z=1}^{k} e^{V(c_{i,j+z})|_{x_i \in c_{i,j+z}}/(2\tau)}} \tag{2.2.3}$$

where $s = 1, 2 \cdots, k$. After the cell is chosen, an action is taken by moving from the current dimensional state to a random point of the selected cell. The step length can be denoted as η, $\eta = (\xi + \zeta) w_{\mathrm{c},i}$, where ξ denotes the distance (in the form of number of cells) between the current cell and the selected cell; ζ is a random number and $\zeta \in (0, 1]$. With this step length, current dimensional state x_i moves to a new point of the dimension, x'_i, where $x'_i = x_i - \eta$ if L_1 is selected, or $x'_i = x_i + \eta$ if L_2 is chosen, as shown in Fig. 2.2.

Reinforcement scheme $\mathcal{T}$, adopted by learning automata to calculate the two probabilities, makes good use of the derived path values and the memory of cell values, as described in (2.2.2) and (2.2.3). $\mathcal{T}$ ensures an efficient search, while keeping a balance between the ability of exploration and exploitation.

Reinforcement signal

In order to evaluate the effectiveness of the selected action, it is necessary to observe the reinforcement signal of the search action. As one automaton works, state X could be denoted as $X(x_i)$, which implies that only dimensional state x_i impacts the state and all the other elements of X are unchanged. Once dimensional state x_i moves to x'_i, the ith element of the current state, $X(x_i)$ is replaced by $X(x'_i)$. The fitness value of state $X(x'_i)$ can be obtained by applying $X(x'_i)$ in the objective function of the minimisation problem, $F(\cdot)$. It should be noticed that maximisation problems can be easily transferred to minimisation problems by adding a negative sign to the fitness value. Suppose dimensional state x'_i locates in cell $c_{i,j}$. Then a reinforcement signal is assigned to cell $c_{i,j}$ by comparing the fitness value of $X(x'_i)$ and that of the latest best solution, denoted by X_{best}, which has the minimum fitness value found so far in the learning process. The reinforcement signal is obtained according to the following rule:

$$r(X(x'_i)) = \begin{cases} 1 & \text{if } F(X(x'_i)) \leq F(X_{\text{best}}) \\ 0 & \text{otherwise} \end{cases} \tag{2.2.4}$$

It can be seen that $r = 1$ represents a favorable response; $r = 0$ is unfavorable response and no effort has been made by current dimension state. The reinforcement signal, $r(X(x'_i))$, is used to update the cell value as discussed in Section 2.2.1.

The relationship between state X and the latest best solution, X_{best}, can be described by the following rule, which is used to update X_{best}:

$$X_{\text{best}} \leftarrow \begin{cases} X(x'_i),\, X(x'_i) = [x_1, \ldots, x_{i-1}, x'_i, x_{i+1}, \ldots, x_N] & \text{if } \ r = 1 \\ X_{\text{best}} & \text{otherwise} \end{cases} \tag{2.2.5}$$

Evaluation of cell values

As one of the major functions of memory, updating cell values relies on the reinforcement signal of current dimensional state x_i, $i.e.$ $r(X(x_i))$, and the weighted estimates of the values of possible paths, $i.e.$ the two path values, $L_1(x_i)$ and $L_2(x_i)$, obtained according to (2.2.1). The value of cell $c_{i,j}$, where the current dimensional state x_i locates, is updated as follows:

$$\begin{aligned} &V(c_{i,j})|_{x_i \in c_{i,j}} \leftarrow r(X(x_i)) + \alpha V(c_{i,j})|_{x_i \in c_{i,j}} + \\ &(1-\alpha)\big((1-\lambda_2)L_{\max}(x_i) + \lambda_2 L_{\min}(x_i)\big) \end{aligned} \tag{2.2.6}$$

where $L_{\max}(x_i)$ and $L_{\min}(x_i)$ are the two estimated path values at dimensional state x_i, while satisfying $L_{\max}(x_i) = \max\{L_1(x_i), L_2(x_i)\}$ and $L_{\min}(x_i) = \min\{L_1(x_i), L_2(x_i)\}$. It can be seen that the cell value accumulates the effort ($i.e.$ reinforcement signal) that has been made by the dimension states located in the cell. Additionally, the cell value also reveals the neighboring environment of dimension state x_i, as $L_{\max}(x_i)$ and $L_{\min}(x_i)$ represent the potential of finding the better solutions on both sides of x_i. Between the two path values, $L_{\max}(x_i)$ has greater influence on the cell value than $L_{\min}(x_i)$, therefore parameter λ_2 should be given such that $(1-\lambda_2) > \lambda_2$. Weights α and $(1-\alpha)$ given in (2.2.6) represent the influence of previous estimates and neighboring environment ($i.e.$ path values) on the new estimate, respectively. If the reinforcement signal obtained from the same cell remains unfavorable continuously for several times, the cell value will be decreased, which implies that the cell gradually loses its favour of being selected in the future search.

The cell values are to memorize the outcome of searching on the cells when they were visited in the past. Being regarded as past experience, the cell values are used to guide the future search. This concept of cell values is inspired from state values adopted in reinforcement learning [82], and is the main characteristic that differentiates FOLA from other traditional learning automata methods. Hence, FOLA can also be considered as the method in between learning automata and reinforcement learning, rather than a conventional learning automata method [84].

Perturbation operation

In order to increase the diversity and escape from local optima, different perturbations are introduced to the dimensional states of X simultaneously, according to the following rule:

$$X \leftarrow X + \Delta, \ \Delta = [\Delta_1, \dots, \Delta_i, \dots, \Delta_N] \tag{2.2.7}$$

where $\Delta_i = \text{sign}(\kappa)\zeta(x_{\max,i} - x_{\min,i})$, ζ is a random variable and $\zeta \in [0, \frac{k}{D}]$. The sign function is used to choose the moving direction of x_i. Suppose x_i is located in cell $c_{i,j}$, then the input to the sign function is the subtraction of the two adjacent cell values of $c_{i,j}$, that is, $\kappa = (V(c_{i,j+1}) - V(c_{i,j-1}))$. If $\kappa \geq 0$, $\text{sign}(\kappa) = 1$, otherwise,

$\mathrm{sign}(\kappa) = -1$. It can be seen that x_i moves towards the direction in which the adjacent cell has a larger cell value. Once completing the perturbation operation, X_{best} is updated according to the following equation:

$$X_{\mathrm{best}} \leftarrow \begin{cases} X & \text{if } F(X) < F(X_{\mathrm{best}}) \\ X_{\mathrm{best}} & \text{otherwise} \end{cases} \tag{2.2.8}$$

2.2.2 The pseudocode of FOLA

The computation of FOLA includes a certain number of episodes and each episode consists of two stages:

- In the first stage, starting with the first state, by selecting automaton i, at dimensional state x_i, a search is undertaken by calculating the values of two possible paths using the cell values obtained in the previous iterations, as given in (2.2.1), and then moving x_i towards a path, which is selected according to the probability calculated using the path values, as in (2.2.2), with a step length η, which is determined by the probability calculated from the cell values of the cells locating on the path, as in (2.2.3). A reinforcement signal of dimensional state x_i is generated using (2.2.4). The best solution, X_{best}, is updated according to (2.2.5). Then the estimates of the path values and the reinforcement signal are used to update the cell value of $c_{i,j}$, in which dimensional state x_i locates, using (2.2.6). In this case the fitness value of the objective function is calculated once, and the cell value of $V(c_{i,j})|_{x_i \in c_{i,j}}$ is updated to replace the cell value previously stored when any dimensional state located in cell $c_{i,j}$ was visited. If the reinforcement signal is not equal to 1, the current iteration ends and the next automaton is selected in order for FOLA computation to continue. Otherwise searching on this path is regarded worthy, as the cells locating on this path could be very sensitive and need more exploitation. In order to further exploit the potential of the dimensional states located on this path, the search action is taken continuously for I_{emax} times in the same way presented in (2.2.1), (2.2.2), (2.2.3), (2.2.4), (2.2.5) and (2.2.6). An iteration completes after the exploitation, which has updated X_{best} and all values of the cells visited, before the next automaton is selected.

- After all automata are selected sequentially once, the episode goes to its second stage, in which all the automata take actions simultaneously according to (2.2.7). Then X_{best} is updated using (2.2.8).

The FOLA computation proceeds in episodes, until a given maximum number of objective function evaluations, N_{femax}, is reached. The pseudocode of FOLA is listed in Algorithm 1.

2.2.3 Search behaviors of FOLA

The search behaviors conducted by FOLA enable the automata to find accurate solutions efficiently. To better explain FOLA's search behaviors, an example is given in Fig. 2.3. Let the initial $F(X_{\text{best}})$ be an infinite value. Assume an action is taken and the dimensional state moves to point x^1. Since the fitness value is improved ($i.e.$ $F(x^1) < F(X_{\text{best}})$), $r = 1$ and $F(X_{\text{best}}) = F(x^1)$ are set according to (2.2.4) and (2.2.5) respectively. Since $r = 1$, the cell value of x^1 increases according to (2.2.6). Then, as given in line 13 of Algorithm 1, I_{emax} actions will be performed around point x^1. After the loop of lines 13-16 is completed, another point is randomly selected as the new dimensional state, as shown in line 5.

It should be mentioned that the cell, which the randomly selected dimensional state locates in, is uniformly selected, in order to ensure that all cells have equal probabilities of being chosen. If the selected point locates in different places, different search behaviors will be conducted by the automata as follows:

- Assume the next dimensional state moves to point x^2. According to (2.2.1), the path value of the left path is larger than that of the right path, which leads to the left path being selected with a larger probability due to (2.2.2). With this searching rule, it is more likely that the search will be finally attracted to cell C, according to (2.2.3).

- If the next dimensional state locates at point x^4, the automaton will take only one action, then it stop searching around this area, due to $F(x^4) > F(X_{\text{best}})$, as shown in line 10 of Algorithm 1.

Algorithm 1 The pseudocode of FOLA

```
1: Initialise a state $X = [x_1, \cdots, x_i, \cdots, x_N]$ arbitrarily; set $X_{\text{best}} = X$; and initialise cell values $V$ arbitrarily at the range of [0 1]
2: Set $N_{\text{fe}} = 0$
3: **while** $N_{\text{fe}} \leq N_{\text{femax}}$ **do**
4:     **for** $i = 1$ to $N$ **do**
5:         Select a dimensional state $x_i$ randomly
6:         Estimate path values $L_l(x_i)$, where $l = 1, 2$ according to (2.2.1)
7:         Take a search action to move dimensional state $x_i$ to $x_i - \eta$ on path $L_1$ ($x_i \leftarrow x_i - \eta$) or $x_i + \eta$ on path $L_2$ ($x_i \leftarrow x_i + \eta$) according to (2.2.2) and (2.2.3)
8:         Calculate reinforcement signal $r(X(x_i))$ according to (2.2.4), set $N_{\text{fe}} = N_{\text{fe}} + 1$, and update $X_{\text{best}}$ according to (2.2.5)
9:         Update cell value $V(c_{i,j})|_{x_i \in c_{i,j}}$ according to (2.2.6), where $j = \text{floor}((x_i - x_{\text{min},i})/w_{\text{c},i})$, and $\text{floor}(y)$ denotes the greatest integer less than the real value $y$
10:        **if** $r(X(x_i)) \neq 1$ **then**
11:            go to step 4
12:        **end if**
13:        **for** $I_{\text{e}} = 1$ to $I_{\text{emax}}$ **do**
14:            Use (2.2.1) and (2.2.2) to estimate path values and path probabilities, and generate $\eta$ using (2.2.3), in order to take a search action with an alternative step length
15:            Use (2.2.4), (2.2.5) and (2.2.6) again to calculate reinforcement signal and update $X_{\text{best}}$ and $V(c_{i,j})|_{x_i \in c_{i,j}}$ respectively
16:        **end for**
17:    **end for**
18:    Use (2.2.7) to add perturbations to the dimensional states of $X$
19:    Use (2.2.8) to update $X_{\text{best}}$
20: **end while**
```

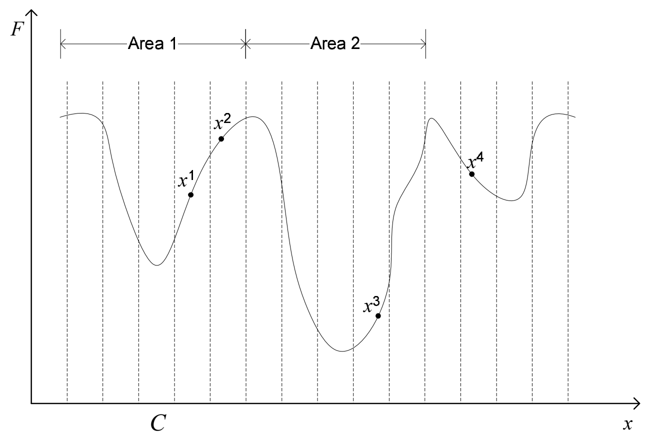


Figure 2.3: The illustration of the search behavior of automata

- If the next dimensional state locates at point x^3, since $F(x^3) < F(X_{\text{best}})$, more search will be performed around x^3 (line 13 of Algorithm 1). Subsequently, the cell values of Area 2 will increase, which leads to more search around Area 2 due to (2.2.1), $i.e.$ the larger the cell values are, the more exploitation the corresponding cells can attract. In this way, once the new promising Area 2 is discovered, it will attract more exploration than Area 1 for the subsequent search. At the same time, the cell value of C will decrease gradually due to parameter α in (2.2.6). This search behaviour can increase the efficiency of the search.

FOLA can find promising areas efficiently, thanks to the aforementioned adaptive search behaviors and the benefit of memorising historical estimate cell values which are used for future search guidance.

2.3 Compared with Classical EAs

2.3.1 Benchmark functions

There are various benchmark functions used for comparison in the literature of optimisation [1] [85]. According to No Free Lunch theorem, "for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class" [86]. We are not motivated to develop an algorithm which can solve all functions, but are motivated to develop an algorithm that can solve a class of benchmark functions in terms of accuracy and efficiency. Here, 13 challenging benchmark functions selected from the set of 23 standard benchmark functions [1] are used in the simulation. Appendixes A.1 and A.2 list the 13 benchmark functions used in this section, including uni-modal functions ($F_1 \sim F_7$) and multi-modal functions ($F_8 \sim F_{13}$), where the number of dimensions, N, the search range of each dimension and the minimum value of each function are indicated. Among them, functions $F_8 \sim F_{13}$ are regarded as difficult functions to be optimised since a large number of local optima is found [87], and the number of the local optima increases exponentially as the dimensionality of the problem increases.

2.3.2 Evaluation on 30-dimensional functions

EAs involved in comparison

In solving the 30-dimensional benchmark functions, $F_1 \sim F_{13}$, FOLA is compared with the following algorithms:

1) Evolutionary Programming based on Reinforcement Learning (RLEP) [15]
2) Mixed Strategy Evolutionary Programming (MSEP) [14]
3) Particle Swarm Optimisation (PSO) [46]
4) Genetic Algorithm (GA) [16]
5) Conventional Evolutionary Programming (CEP) [17] [88]
6) Fast Evolutionary Programming (FEP) [1]
7) Conventional Evolution Strategies (CES) [89]
8) Fast Evolution Strategies (FES) [90]

The parameters setting of RLEP and MSEP refers to [15] and [14] respectively. The parameters of CEP and FEP follow the recommendation in [1], and those of CES and FES refer to [90]. The parameters of PSO are given as follows: both acceleration factors c_1 and c_2 are 2.0; a decaying inertia weight ω starting at 0.9 and ending at 0.4 is used; the population size is 50. GA is real-coded with intermediate crossover and Guaussian mutation. Its population size is 50, and the reproduction function is conducted using uniform stochastic selection. To improve the search performance of GA on multi-modal optimisation problems, subpopulation is used, and the migration rate is set to 0.1. All the control parameters, *e.g.* mutation rate and crossover rate, $etc.$, refer to default values as recommended in [91]. In this section, the experimental results of RLEP and MSEP are adopted directly from references [15] and [14] respectively, and the results of the rest algorithms, except for FOLA, refer to [56]. The parameters setting of FOLA, except for w_{c} and N_{femax}, does not affect the performance substantially when FOLA is applied to solve different optimisation problems, therefore the best set of these parameters is chosen empirically and preset as follows: $\tau = 0.2$, $k = 4$, $\alpha = 0.8$, $\lambda_1 = 0.5$, $\lambda_2 = 0.25$ and $I_{\mathrm{emax}} = 5$. The parameters which need to be tuned carefully in FOLA are w_{c} and N_{femax}, which are listed in Table 2.1. These two parameters can be adjusted according to specific problems, and they will be further discussed in Section 2.3.4. Following the termination criterion given in [1] and [90], the computation process stops when reaching N_{femax}. N_{femax} used by these algorithms in solving the 13 benchmark functions is tabulated in Table 2.1.

Table 2.1: The setting of N_{femax} for 30-dimensional benchmark functions $F_1 \sim F_{13}$

N_{femax}	FOLA (w_{c})	RLEP/MSEP	PSO/GA	CEP/FEP /CES/FES
F_1	150,000 (0.5)	150,000	150,000	150,000
F_2	150,000 (1)	200,000	150,000	200,000
F_3	150,000 (5)	-	150,000	500,000
F_4	150,000 (2)	500,000	150,000	500,000
F_5	150,000 (1)	150,000	150,000	2,000,000
F_6	150,000 (2)	150,000	150,000	150,000
F_7	150,000 (0.1)	300,000	150,000	300,000
F_8	150,000 (5)	900,000	150,000	900,000
F_9	150,000 (0.04)	500,000	150,000	500,000
F_{10}	150,000 (1)	150,000	150,000	150,000
F_{11}	150,000 (5)	200,000	150,000	200,000
F_{12}	150,000 (1)	150,000	150,000	150,000
F_{13}	150,000 (1)	150,000	150,000	150,000

Simulation results

FOLA is tested on the 30-dimensional uni-modal functions, $F_1 \sim F_7$, in comparison with the other eight algorithms. Fifty independent runs of the FOLA were executed. Table 2.2 lists the mean and standard deviation of the fitness values obtained by the nine algorithms and their rank. Among the benchmark functions given in Appendixes A.1 and A.2, F_3 is the only function that references [15] and [14] do not provide any relevant results. It can be seen from Table 2.2 that FOLA generates better results than RLEP on these functions except for F_7; FOLA outperforms MSEP in solving these benchmark functions. In comparison with other algorithms, FOLA has significantly better performance on the seven functions except for function F_3. In solving function F_3, CES has the best performance among the nine algorithms, however, it performs much worse than others in solving the rest functions. Additionally, the standard deviation obtained by FOLA is much smaller than that of other algorithms when solving these functions except for F_3. On average, FOLA performs better than other algorithms with respect to the accuracy of the results and the reliability which is revealed by its smaller standard deviation over different independent runs.

Functions $F_8 \sim F_{13}$ $(N = 30)$ are regarded as difficult functions to be optimised since a large number of local optima is found [87]. Fifty independent runs of the FOLA were executed. The experimental results, including mean, standard deviation and rank, are listed in Table 2.3. It is clear to see that FOLA markedly outperforms other algorithms, especially for the functions F_9 and $F_{11} \sim F_{13}$. For many practical applications, $e.g.$ the optimisation of power dispatch in power systems, etc, each evaluation of a fitness value needs a long time. Therefore, reducing the number of function evaluations, while achieving accurate results simultaneously, is greatly demanded in practical applications, especially for those on-line applications where the time for finding global optimum is limited. Regarding to this aspect, FOLA greatly presents its superiority over RLEP and MSEP: N_{femax} used by RLEP and MSEP on function F_8 is five times more than that of FOLA; RLEP and MSEP use more than three times of the N_{femax} used by FOLA in solving function F_9.

Table 2.2: Comparison among FOLA and the other eight algorithms on 30-dimensional benchmark functions $F_1 \sim F_7$: Average fitness value / (Standard deviation) / (**Rank**)

	F_1	F_2	F_3	F_4	F_5	F_6	F_7
FOLA	3.50×10^{-40}	4.17×10^{-26}	6.13×10^{-1}	1.69×10^{-6}	3.06×10^{-19}	0	1.30×10^{-4}
	(2.09×10^{-40})	(1.56×10^{-26})	(4.57×10^{-1})	(1.62×10^{-6})	(5.34×10^{-19})	(0)	(2.60×10^{-3})
	(1)	**(1)**	**(6)**	**(1)**	**(1)**	**(1)**	**(2)**
RLEP	1×10^{-4}	2.5×10^{-4}	-	4.3×10^{-4}	16.7	0	1.0×10^{-4}
	(1.1×10^{-5})	(1.3×10^{-5})	-	(5.2×10^{-4})	(20.2)	(0)	(8.5×10^{-3})
	(4)	**(3)**		**(2)**	**(5)**	**(1)**	**(1)**
MSEP	1×10^{-4}	4.1×10^{-4}	-	2.7×10^{-2}	29.2	0	6.1×10^{-3}
	(1.3×10^{-5})	(2.1×10^{-4})	-	(1.7×10^{-2})	(24.9)	(0)	(1.7×10^{-3})
	(4)	**(4)**		**(4)**	**(6)**	**(1)**	**(3)**
GA	3.17	5.77×10^{-1}	9749.91	7.96	338.56	3.70	0.10
	(1.66)	(1.31×10^{-1})	(2594.96)	(1.51)	(361.50)	(1.95)	(3.62×10^{-2})
	(8)	**(9)**	**(7)**	**(9)**	**(9)**	**(3)**	**(9)**
PSO	3.69×10^{-37}	2.92×10^{-24}	1.20×10^{-3}	0.41	37.36	0.15	9.90×10^{-3}
	(2.46×10^{-36})	(1.14×10^{-23})	(2.11×10^{-3})	(0.25)	(32.14)	(0.42)	(3.54×10^{-2})
	(2)	**(2)**	**(2)**	**(7)**	**(8)**	**(2)**	**(5)**
FEP	5.7×10^{-4}	8.1×10^{-3}	1.6×10^{-2}	0.3	5.06	0	7.6×10^{-3}
	(1.3×10^{-4})	(7.7×10^{-4})	(1.4×10^{-2})	(0.5)	(5.87)	(0)	(2.6×10^{-3})
	(7)	**(6)**	**(4)**	**(5)**	**(2)**	**(1)**	**(4)**
CEP	2.2×10^{-4}	2.6×10^{-3}	5.0×10^{-2}	2.0	6.17	577.76	1.8×10^{-2}
	(5.9×10^{-4})	(1.7×10^{-4})	(6.6×10^{-2})	(1.2)	(13.61)	(1,125.76)	(6.4×10^{-3})
	(5)	**(5)**	**(5)**	**(8)**	**(3)**	**(5)**	**(7)**
FES	2.5×10^{-4}	6.0×10^{-2}	1.4×10^{-3}	5.5×10^{-3}	33.28	0	1.2×10^{-2}
	(6.8×10^{-4})	(9.6×10^{-3})	(5.3×10^{-4})	(6.5×10^{-4})	(43.13)	(0)	(5.8×10^{-3})
	(6)	**(8)**	**(3)**	**(3)**	**(7)**	**(1)**	**(6)**
CES	3.4×10^{-5}	2.1×10^{-2}	1.3×10^{-4}	0.35	6.69	411.16	3.0×10^{-2}
	(8.6×10^{-6})	(2.2×10^{-3})	(8.5×10^{-5})	(0.42)	(14.45)	(695.35)	(1.5×10^{-2})
	(3)	**(7)**	**(1)**	**(6)**	**(4)**	**(4)**	**(8)**

Table 2.3: Comparison among FOLA and the other eight algorithms on 30-dimensional benchmark functions $F_8 \sim F_{13}$: Average fitness value / (Standard deviation) / (**Rank**)

	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}
FOLA	-12569.5	6.33×10^{-11}	1.43×10^{-5}	1.76×10^{-10}	1.02×10^{-11}	1.13×10^{-11}
	(1.73×10^{-4})	(3.90×10^{-11})	(3.14×10^{-6})	(7.65×10^{-11})	(9.14×10^{-12})	(1.18×10^{-11})
	(**1**)	(**1**)	(**1**)	(**1**)	(**1**)	(**1**)
RLEP	-12569.5	1.0×10^{-5}	1.6×10^{-3}	5.6×10^{-4}	6.9×10^{-7}	1.1×10^{-5}
	(0)	(2.3×10^{-6})	(4.4×10^{-4})	(1.2×10^{-3})	(1.5×10^{-7})	(1.3×10^{-6})
	(**1**)	(**2**)	(**3**)	(**2**)	(**2**)	(**2**)
MSEP	-12569.48	2.5×10^{-5}	1.7×10^{-3}	8.5×10^{-4}	7.5×10^{-7}	1.2×10^{-5}
	(3.4×10^{-4})	(2.1×10^{-5})	(4.3×10^{-4})	(2.3×10^{-3})	(4.0×10^{-7})	(1.1×10^{-5})
	(**2**)	(**3**)	(**4**)	(**3**)	(**3**)	(**3**)
GA	-12566.1	0.65	0.87	1.00	4.36×10^{-2}	0.17
	(2.11)	(0.36)	(0.28)	(6.75×10^{-2})	(5.06×10^{-2})	(7.07×10^{-2})
	(**3**)	(**6**)	(**7**)	(**9**)	(**7**)	(**7**)
PSO	-9659.7	20.79	1.34×10^{-3}	0.23	3.95×10^{-2}	5.05×10^{-2}
	(463.78)	(5.94)	(4.24×10^{-2})	(0.44)	(9.14×10^{-2})	(0.57)
	(**6**)	(**7**)	(**2**)	(**7**)	(**6**)	(**6**)
FEP	-12554.5	4.6×10^{-2}	1.8×10^{-2}	1.6×10^{-2}	9.2×10^{-6}	1.6×10^{-4}
	(52.6)	(1.2×10^{-2})	(2.1×10^{-2})	(2.2×10^{-2})	(6.14×10^{-5})	(7.3×10^{-5})
	(**5**)	(**4**)	(**6**)	(**4**)	(**4**)	(**5**)
CEP	-7917.1	89.0	9.2	8.6×10^{-2}	1.76	1.4
	(634.5)	(23.1)	(2.8)	(0.12)	(2.4)	(3.7)
	(**7**)	(**9**)	(**9**)	(**6**)	(**9**)	(**9**)
FES	-12556.4	0.16	1.2×10^{-2}	3.7×10^{-2}	2.8×10^{-2}	4.7×10^{-5}
	(32.53)	(0.33)	(1.8×10^{-3})	(5.0×10^{-2})	(8.1×10^{-11})	(1.5×10^{-5})
	(**4**)	(**5**)	(**5**)	(**5**)	(**5**)	(**4**)
CES	-7549.9	70.82	9.07	0.38	1.18	1.39
	(631.39)	(21.49)	(2.84)	(0.77)	(1.87)	(3.33)
	(**8**)	(**8**)	(**8**)	(**8**)	(**8**)	(**8**)

2.3.3 Evaluation on 300-dimensional functions

In the literature, setting benchmark functions in 30 dimensions is commonly used for algorithm comparison. However, many real-world problems usually involve hundreds of variables. Therefore, it is crucial to investigate whether FOLA can be scaled up to handle the optimisation problems which are high-dimensional, $e.g.$ 300. For the multi-modal benchmark functions, $F_8 \sim F_{13}$, the number of their local minima increases exponentially as the number of dimensions increases. In this case, the global minimum value of F_8 should be -125,694.7, while the other functions have the same global minimum values as those given in Appendix A.2.

EAs involved in comparison

In [92] [93], only functions F_1 and F_{10} were tested in 300 dimensions, while the dimensionality of other functions was set to 30. Since the results obtained by some of the EAs presented in Section 2.3.2 in solving the 300-dimensional benchmark functions are not available, the following EAs which have been evaluated on 300-dimensional benchmark functions are adopted for comparison, and their results published in [56] are directly used in this section.

FOLA is compared with the following algorithms:

1) Particle Swarm Optimisation (PSO) [46]

2) Genetic Algorithm (GA) [16]

3) Evolutionary Programming (EP) [88] [94]

4) Evolution Strategies (ES) [89]

The parameters setting of PSO, GA and FOLA is the same as that in solving 30-dimensional functions, as given in Section 2.3.2, except for N_{femax}. For EP, the population size and the tournament size for selection are 100 and 10 respectively. The initial standard deviation of EP is 3.0, and its parameters setting is based on [88] and [94]. ES used in our experiments is a state-of-the-art (μ, λ)-ES. The population, μ, is 30 and the offspring number, λ, is 200. A standard deviation of 3.0 is adopted. A global intermediate recombination [89] is also employed in ES. N_{femax}= 2,000,000 is used in FOLA for solving the 300-dimensional benchmark functions, while N_{femax}=3,750,000 is set for the other four algorithms.

Simulation results

The 300-dimensional multi-modal benchmark functions, $F_8 \sim F_{13}$, are extremely complex with a huge number of local optima to find. Five independent runs of the FOLA were executed. Table 2.4 lists the mean of the fitness values and the rank of the five algorithms, and the standard deviation obtained by FOLA. The results of the algorithms, except for FOLA, are directly adopted from reference [56]. Note that the standard deviation of the algorithms (expect for FOLA) is not provided, as reference [56] reported only the mean of fitness values in solving 300-dimensional functions. It can be seen that FOLA markedly outperforms the other algorithms on these functions with respect to the average fitness value. In terms of computation efficiency, N_{femax} used by FOLA is much smaller than that of other algorithms which need a large N_{femax} to obtain the results given in Table 2.4. The superiority of FOLA over other algorithms is greatly enhanced by its accurate results and the use of much smaller N_{femax}.

Table 2.4: Comparison among FOLA, GA, PSO, EP and ES on 300-dimensional benchmark functions $F_8(x) \sim F_{13}(x)$: Average fitness value / (Standard deviation) / (**Rank**)

	F_8	F_9	F_{10}	F_{11}	F_{12}	$F_{13}(x)$
FOLA	-125,694.69	3.51×10^{-7}	1.47×10^{-7}	7.02×10^{-8}	8.08×10^{-9}	4.91×10^{-9}
	(0.19)	(5.67×10^{-8})	(1.65×10^{-8})	(3.60×10^{-8})	(3.79×10^{-9})	(4.23×10^{-10})
	(**1**)	(**1**)	(**1**)	(**1**)	(**1**)	(**1**)
GA	-117,275.3	121.3	6.24	0.37	52.82	178.34
	-	-	-	-	-	-
	(**2**)	(**2**)	(**4**)	(**2**)	(**2**)	(**4**)
PSO	-87,449.2	427.1	3.9540×10^{-6}	1.81	14.56	549.2
	-	-	-	-	-	-
	(**3**)	(**4**)	(**2**)	(**3**)	(**4**)	(**2**)
EP	-78,311.9	383.3	0.2946	2.82×10^{-2}	39.3	738.2
	-	-	-	-	-	-
	(**4**)	(**3**)	(**3**)	(**4**)	(**3**)	(**3**)
ES	-66,531.3	583.2	9.6243	0.16	3093.2	2123.2
	-	-	-	-	-	-
	(**5**)	(**5**)	(**5**)	(**5**)	(**5**)	(**5**)

2.3.4 Discussion

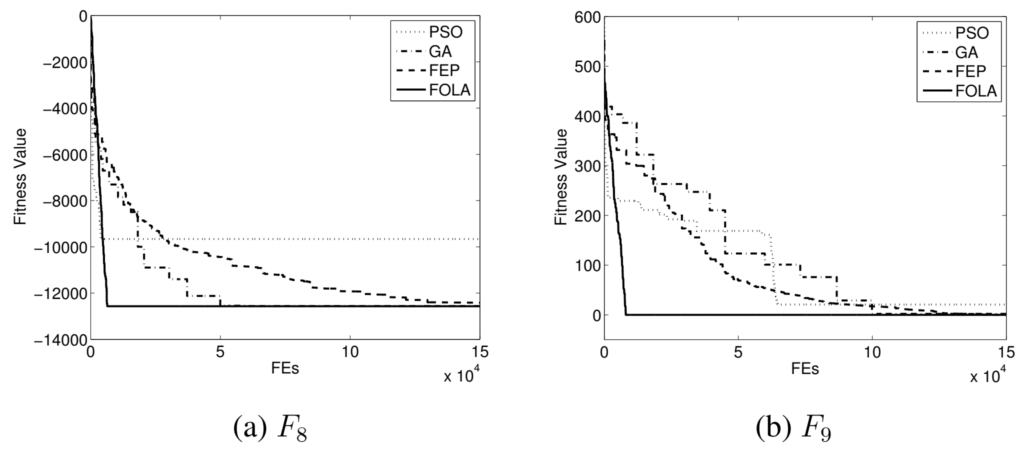
Convergence characteristics

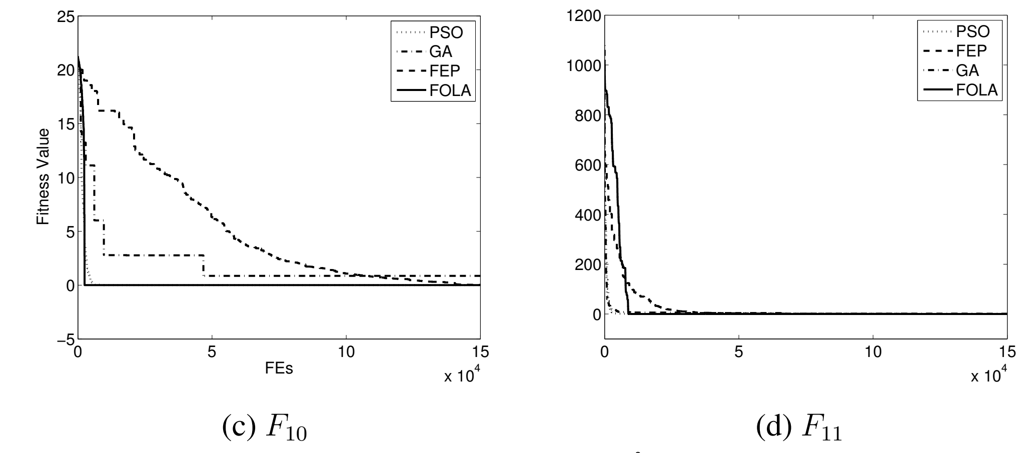
The convergence comparison among FOLA and other EAs is carried out on 30-dimensional multi-modal benchmark functions, $F_8(x) \sim F_{13}(x)$. It should be mentioned that the experimental results of the algorithms (except for FOLA) given in Section 2.3.2 and 2.3.3 are directly adopted from the published papers. However, the convergence comparison requires the evolvement of the fitness values recorded in the optimisation process, rather than the final best fitness value. Since the codes and toolbox of algorithms FEP, PSO and GA are publicly available, they are used to compare with FOLA in terms of convergence rates. The parameters setting of FEP is listed as follows: the population size is set to $\mu = 100$; the tournament size $q = 10$ is preset for selection; the initial $\eta = 3.0$; the initial population is generated uniformly at random in the search range. The parameters of PSO are set as follows: both acceleration factors c_1 and c_2 are set to 2.0; a decaying inertia weight ω starting at 0.9 and ending at 0.4 is used; the population size is 50. The parameters setting of GA is given as follows: the population size is set to 50; the mutation rate is 0.05; the crossover rate is 0.8; the reproduction function applies uniform stochastic selection.

The convergence rates of PSO, GA, FEP and FOLA are presented respectively in Fig. 2.4. GA has the slowest convergence rate among the four algorithms. It can be seen that GA has been given enough FEs to find the optimal solution, as there is no improvement of the convergence rate in the late stage of the optimisation. FEP outperforms GA, but it has somewhat slower convergence rate when compared with PSO and FOLA. At the beginning of the optimisation process, PSO converges fastest among the four algorithms. However, around 10×10^4 FEs, PSO is surpassed by FEP when solving functions F_{10}, F_{11} and F_{12}. Although FEP has a slower convergence rate than PSO at the beginning of optimisation, it maintains a consistent convergence rate throughout the evolution process, and overtakes PSO at the late stage of the optimisation process. As for FOLA, it converges slower than PSO at the beginning, due to the reason that a certain number of FEs is required to build up the memory. However, it surpasses the other three algorithms quickly.

Computation time

FOLA is compared with FEP, PSO and GA, with respect to the computation time, on the 30-dimensional benchmark functions $F_8(x) \sim F_{13}(x)$. The parameters setting of these algorithms follows those given in Section 2.3.4. These algorithms were implemented using MATLAB R2008b in a PC which has a CPU of Intel(R) Core(TM)2 Duo 3.33GHz and a memory of 2G. Table 2.5 lists the average compu-


(a) F_8 (b) F_9


(c) F_{10} (d) F_{11}

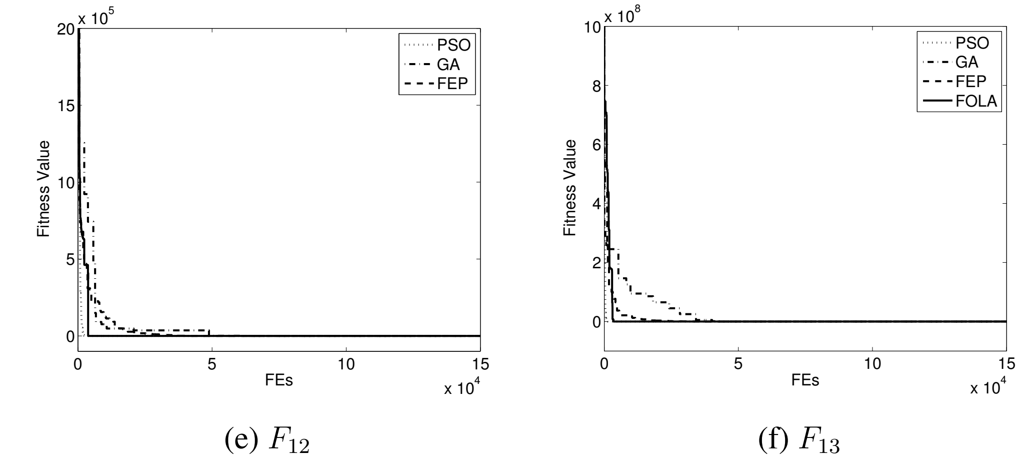

(e) F_{12} (f) F_{13}

Figure 2.4: The comparison of convergence rates among PSO, GA, FEP and FOLA on the 30-dimensional benchmark functions $F_8(x) \sim F_{13}(x)$

tation time consumed by the four algorithms when solving functions $F_8(x) \sim F_{13}(x)$ respectively. It can be seen that GA consumes much longer time than the other three algorithms on these six functions. PSO requires less computation time than FEP on these functions except for F_9 and F_{10}. FOLA consumes the shortest computation time among the four algorithms on these functions except for function F_{10}. On average, FOLA outperforms the other algorithms in terms of computation time.

Table 2.5: Computation time (s) consumed by PSO, GA, PSO and FOLA when solving the 30-dimensional benchmark functions $F_8(x)$∼$F_{13}(x)$

	F_8	F_9	F_{10}	F_{11}	F_{12}	$F_{13}(x)$
FOLA	4.97	6.17	9.98	7.25	9.43	9.39
PSO	9.48	9.02	10.03	10.31	12.79	12.98
GA	63.58	62.71	63.57	64.69	65.97	65.25
FEP	10.39	8.76	9.93	11.60	15.90	15.95

Parameters w_c and N_{femax}

FOLA has two parameters, w_c and N_{femax}, which should be set up with some basic knowledge of the complexity of optimisation problems, as they are related to the accuracy of the solutions and the computation efficiency of FOLA. Investigation on the sensitivity of optimisation results to parameters w_c and N_{femax} has been undertaken on F_9, since it has been noted that the results obtained from this function are more sensitive than those obtained from other functions when the parameters are set to different values. FOLA was performed for 50 runs for each set of parameters, in order to generate the mean of the minimum fitness values for comparison.

Parameter w_c is related to the resolution of dividing the dimensions to cells and parameter N_{femax} is used to terminate the computation process. Parameter w_c should be set properly to ensure the accuracy of the results found by FOLA. Under the condition that N_{femax} is large enough, setting smaller w_c can yield more accurate solutions. However, a small w_c requires a large N_{femax} to converge. Therefore, w_c is the parameter to tradeoff between the accuracy and computation load. Table 2.6 lists the results obtained using a range of different w_c and N_{femax}, which are consistent with the analysis. Therefore, both N_{femax} and w_c should be set properly for a specific

Table 2.6: The average fitness values obtained with different w_c and N_femax

w_c	1	0.4	0.04	0.004	0.0004
N_femax	4,000	65,000	150,000	150,000	150,000
F_9	3.85	0.31	6.33×10^{-11}	9.45×10^{-9}	182.47

application so as to ensure the quality of solutions and the efficiency of computation.

2.4 Compared with Recently-proposed EAs

2.4.1 Benchmark functions

FOLA can efficiently solve the problems given in Appendixes A.1 and A.2. In order to investigate the potential of FOLA in solving other problems and fully evaluate the performance of FOLA without a biased conclusion, another class of benchmark functions, which are rotated and shifted, are also employed for comparison. Appendix A.3 lists the rotated benchmark functions employed in this section, together with the number of dimensions, N, the search range of each dimension and the minimum value (F_min) of each function. For these functions, parameters O, M and f_bias denote the shift, rotation and bias respectively.

For this type of problems, FOLA adds the concept of covariance matrix adaption C [95] in the perturbation operation. C is to approximate inverse Hessian matrix and thus guide the search according to the contour lines of the objective function. Based on the mean of previous X_best (denoted as m), a new step is taken through a perturbation that is generated by $\mathcal{N}(0, C)$, the multivariate normal distribution with zero mean and covariance matrix C. The covariance matrix is updated according to the following equation:

$$C \leftarrow (1 - c_\mathrm{cov})C + c_\mathrm{cov} Y_i Y_i^\mathrm{T} \tag{2.4.1}$$

where c_cov is the learning rate of the covariance matrix; $Y_i = (X_\mathrm{best} - m)/\sigma$; and σ is the step-size. The setting of these parameters can refer to [95].

Other parameters of FOLA are set as those given in 2.3.2, with the exception of w_c and N_{femax}. Given the search range of each variable, w_c is automatically determined if D is preset, as aforementioned in Section 2.2. D is set to 20 for this class of benchmark functions. As for N_{femax}, it is set to the same value for all the algorithms, in order to have a fair comparison.

2.4.2 Compared with CLPSO and CPSO

To fully evaluate the performance of FOLA in solving this class of benchmark functions, two popular particle swarm optimisers, Comprehensive Learning Particle Swarm Optimiser (CLPSO) [51] and Cooperative Particle Swarm Optimisation (CPSO) [50], are used for comparison. The parameters setting of CLPSO and CPSO refers to [51] and [50] respectively. N_{femax} is set to 300,000 for algorithms CLPSO, CPSO and FOLA. For the following tables given in this section, the algorithm which performs best in one problem will be highlighted in grey colour. It should be mentioned that if the gap between two algorithms' fitness values is less than 10^{-5}, it will be considered that the two algorithms have the same performance, and two of them will be highlighted if their performance is better than other algorithms.

Table 2.7 lists the mean and standard deviation of the fitness values obtained by the three algorithms over 50 independent runs. In order to further assess the performance of the FOLA in a stochastic search process with a consideration of randomly distributed initial populations, a set of two-tailed t-tests were adopted [1] [96]. The t-test assesses whether the means of two groups of results are statistically different from each other. In this case, the statistical difference of the experimental results between the FOLA and the other two algorithms are measured. It can be seen from Table 2.7 that FOLA performs better than CLPSO and CPSO on functions F_{rs1}, F_{rs2} and $F_{\text{rs5}} \sim F_{\text{rs8}}$, in terms of the average fitness values; FOLA has the same mean value as CLPSO on function F_{rs4}; For functions F_{rs3} and F_{rs9}, FOLA ranks second, as CPSO has a smaller mean value on function F_{rs3} and CLPSO obtains a smaller mean value on function F_{rs9}. For t-test value, it should be mentioned that when the value is negative, it means that FOLA outperforms the corresponding algorithm in terms of both mean and standard deviation; and vice versa. In Table

Table 2.7: Comparison among FOLA, CLPSO and CPSO on 30-dimensional benchmark functions $F_{\text{rs1}} \sim F_{\text{rs9}}$, including Average fitness value, Standard deviation and t-test

Function	Algorithms	Mean	Std.	t-test
F_{rs1}	FOLA	390.1328875	0.727854604	N/A
	CLPSO	395.5523763	5.440959418	-5.407442625
	CPSO	519.2295216	59.26617602	-11.92987495
F_{rs2}	FOLA	-179.9899948	0.016086481	N/A
	CLPSO	4516.288614	2.24692E-12	-1599018.322
	CPSO	4516.288614	8.054E-07	-1599018.32
F_{rs3}	FOLA	-119.1245855	0.092760408	N/A
	CLPSO	-119.0726711	0.058042098	-2.598603419
	CPSO	-119.3901559	0.099867425	10.67187686
F_{rs4}	FOLA	-330.00	9.61078E-13	N/A
	CLPSO	-330	0	5.8312
	CPSO	-329.9997073	0.000950119	-1.6873
F_{rs5}	FOLA	-286.5203526	9.260629049	N/A
	CLPSO	-228.2075732	15.14505415	-17.99194749
	CPSO	78.28110031	110.2864188	-18.05383496
F_{rs6}	FOLA	96.45864503	6.900034916	N/A
	CLPSO	115.1951748	1.613775642	-14.48218587
	CPSO	126.5900006	3.363229076	-21.50014185
F_{rs7}	FOLA	156.5560685	1001.875456	N/A
	CLPSO	49532.24651	10737.6566	-25.07737303
	CPSO	20260.2027	11678.15019	-9.394400183
F_{rs8}	FOLA	-129.1081878	0.249952592	N/A
	CLPSO	-127.8158479	0.218386132	-21.3259
	CPSO	-129.0842205	0.200840166	-0.4094
F_{rs9}	FOLA	-286.510075	0.347176382	N/A
	CLPSO	-287.3025274	0.237869463	10.31355035
	CPSO	-286.4389932	0.305127968	-0.84233251

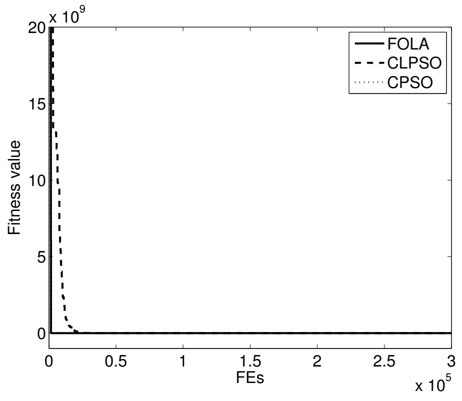
2.7, it can be seen that there are totally 18 t-test values, among which there are 15 negative values and 3 positive values. It suggests that FOLA performs better than the other two algorithms in most of the cases.

Convergence analysis

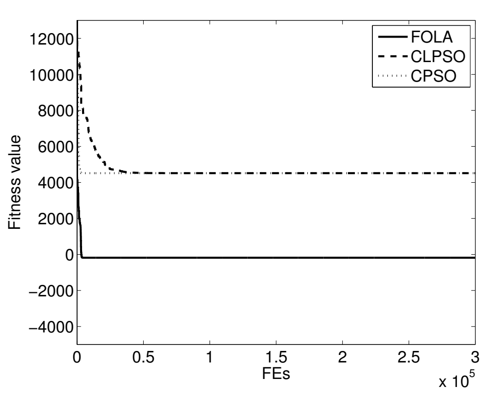
The comparison of convergence rate among the three algorithms is also carried out on the nine multi-modal benchmark functions, by observing the evolvement of the fitness values recorded in the optimisation process. The convergence rates of FOLA, CLPSO and CPSO are presented in Figs. 2.5 and 2.6, where FEs denotes the number of function evaluations. For functions F_{rs1}, F_{rs2}, F_{rs5} and F_{rs7}, it is obvious that FOLA converges much faster than the other two algorithms. As for functions F_{rs3} and F_{rs6}, FOLA has a faster convergence rate in part of the whole convergence line. For instance, in Fig. 2.5(c), FOLA converges faster at the beginning stage of the optimisation process, until at about FEs$=5\times10^4$, FOLA is surpassed by CPSO. In Fig. 2.5(f), FOLA converges faster than the other two algorithms with the exception of a small area, where $27,080 <$FEs$< 51,230$. During the whole optimisation process, FOLA maintains a consistent convergence rate throughout the evolution process, and overtakes CLPSO and CPSO quickly when FEs$= 51,230$. For the rest of the functions, CPSO has a faster convergence rate than FOLA on functions F_{rs4} and F_{rs8}; while CLPSO converges faster on function F_{rs9}. It can be seen from Figs. 2.5 and 2.6 that, on average, CLPSO has a slower convergence rate than FOLA and CPSO.

Computation time

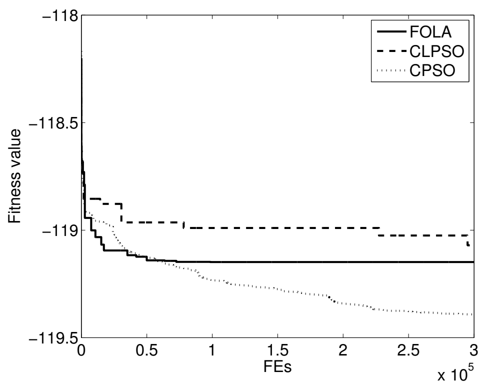
FOLA is also compared with CLPSO and CPSO with respect to computation time, on functions $F_{\mathrm{rs1}}(x)\sim F_{\mathrm{rs9}}(x)$. Table 2.8 lists the average computation time consumed by the three algorithms over 50 independent runs when solving functions $F_{\mathrm{rs1}}(x)\sim F_{\mathrm{rs9}}(x)$ respectively. It can be seen that FOLA requires the shortest computation time among the three algorithms. Taking function F_{rs1} as an example, FOLA saves 42.57% computation time comparing to CLPSO, and 47.14% to CPSO. CPSO consumes longer computation time than FOLA and CLPSO on all of these



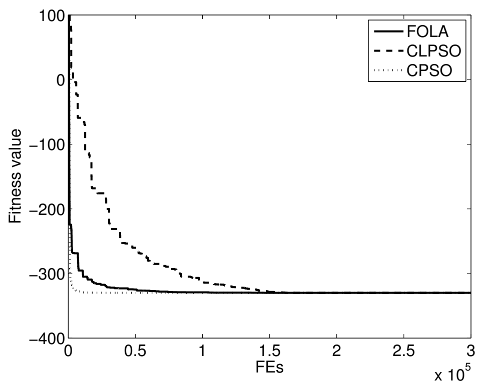
(a) F_{rs1}



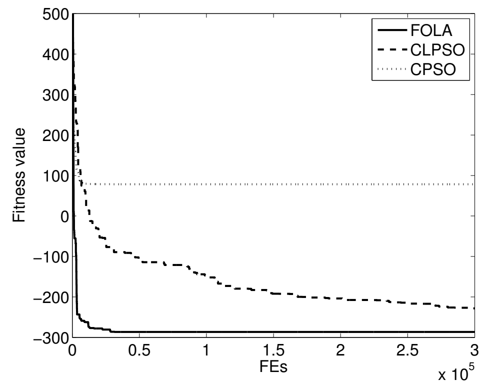
(b) F_{rs2}



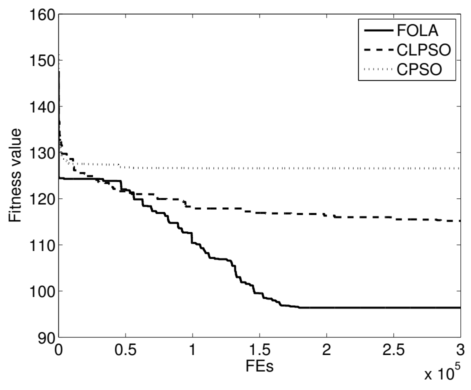
(c) F_{rs3}



(d) F_{rs4}



(e) F_{rs5}

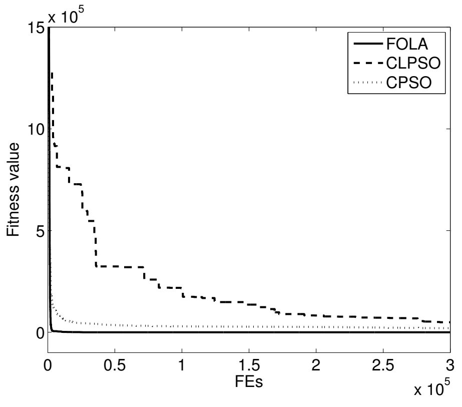


(f) F_{rs6}

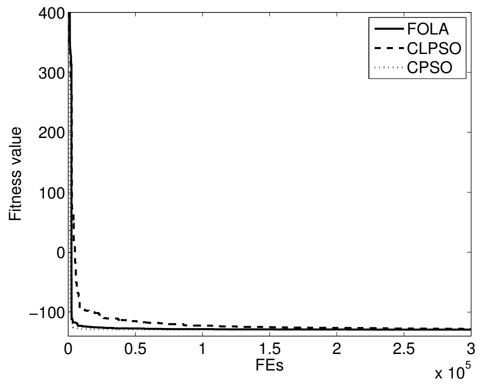
Figure 2.5: The comparison of convergence rates among FOLA, CLPSO and CPSO on the nine benchmark functions, $F_{\mathrm{rs1}}(x) \sim F_{\mathrm{rs6}}(x)$

benchmark functions.

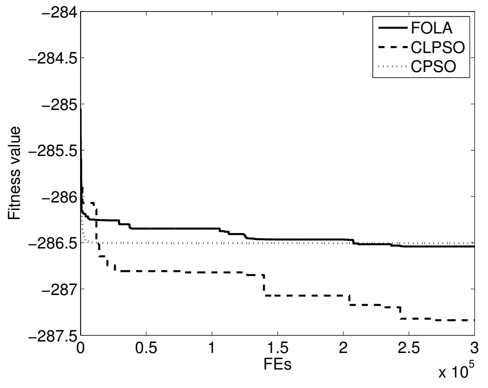
In order to investigate the relationship between the dimensionality of the func-



(a) F_{rs7}



(b) F_{rs8}



(c) F_{rs9}

Figure 2.6: The comparison of convergence rates among FOLA, CLPSO and CPSO on the nine benchmark functions, $F_{\mathrm{rs7}}(x) \sim F_{\mathrm{rs9}}(x)$

Table 2.8: Computation time (s) consumed by FOLA, CLPSO and CPSO when solving the benchmark functions, $F_{\mathrm{rs1}}(x) \sim F_{\mathrm{rs9}}(x)$

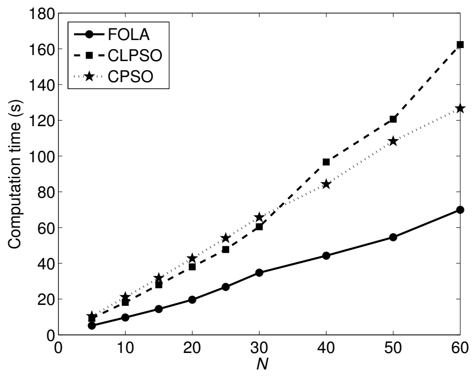
	F_{rs1}	F_{rs2}	F_{rs3}	F_{rs4}	F_{rs5}	F_{rs6}	F_{rs7}	F_{rs8}	F_{rs9}
FOLA	34.73	35.40	30.40	28.72	35.73	184.06	54.88	63.58	76.36
CLPSO	60.47	72.65	66.67	60.95	64.35	263.94	82.52	115.95	127.26
CPSO	65.70	74.44	68.90	67.69	67.54	269.79	84.75	122.11	129.94

tions to be solved and the computation time consumed, the three algorithms are used to solve functions F_{rs1} and F_{rs2}, whose dimensionality N is set to 5, 10, 15, 20, 25, 30, 40, 50 and 60 respectively. For different dimensionality of the functions, N_{femax} used by the algorithms is set to $N \times 10^4$. Fig. 2.7 illustrates the computation time consumed by the three algorithms as the dimensionality increases from 5 to 60. In Fig. 2.7(a), each line can be divided into two segments according to the dimensionality, that is, $5 \leq N \leq 30$ and $40 \leq N \leq 60$. For each segment, an approximate increasing rate can be obtained for each algorithm. When $5 \leq N \leq 30$, the increasing rates of FOLA, CLPSO and CPSO are 1.184, 2.051 and 2.2116 respectively. It can be seen that the increasing rate of FOLA is smaller than those of CLPSO and CPSO. In addition, FOLA saves the computation time by 51.84% on average, in comparison with CPSO; and FOLA saves 45.69% of the computation time compared with CLPSO. When $40 \leq N \leq 60$, the increasing rate of FOLA (1.2845) is still much smaller than those of CLPSO and CPSO, which are 3.2855 and 2.1165 respectively. For this segment, FOLA saves the computation time by 47.37% on average compared with CPSO, and 55.37% compared with CLPSO.

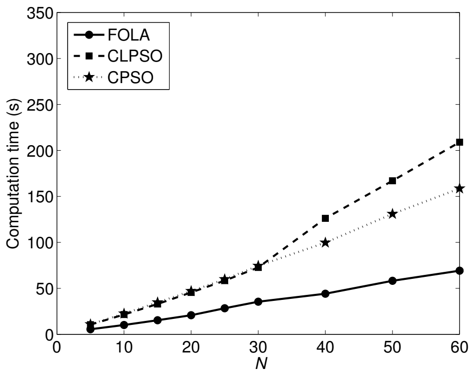
In Fig. 2.7(b), each line can be also divided into two segments according to N. When $5 \leq N \leq 30$, the increasing rates of FOLA, CLPSO and CPSO are 1.1944, 2.4864 and 2.5448 respectively. It can be seen that the increasing rate of FOLA is smaller than half of the increasing rates obtained by CLPSO and CPSO. FOLA saves 53.37% of the computation time on average, in comparison with CPSO; and FOLA saves the computation time by 51.76% compared with CLPSO. When $40 \leq N \leq 60$, the increasing rates of FOLA, CLPSO and CPSO are 1.2515, 4.1402 and 2.9465. For this segment, FOLA saves the computation time by 55.88% on average compared with CPSO, and 65.67% compared with CLPSO.

2.4.3 Compared with GS-SOMA, OLPSO, SOPEN and SamACO

To compare FOLA with newly-proposed optimisation algorithms on the class of benchmark functions given in Appendix A.3, another four algorithms are also employed for comparison in this section. They are Generalized Surrogate Single-Objective Memetic Algorithm (GS-SOMA) [97], Orthogonal Learning Particle Swarm Optimisation (OLPSO) [98], Self-Organizing Potential Field Network (SOPEN) [99] and Sampling Ant Colony Optimisation (SamACO) [100]. These algorithms were used to solve some of the benchmark functions given in Appendix A.3 (or the modified forms of these functions). Their experimental results, which were re-



(a) F_{rs1}



(b) F_{rs2}

Figure 2.7: The comparison of computation time consumed by FOLA, CLPSO and CPSO with respect to different dimensionality, on benchmark functions $F_{\mathrm{rs1}}(x)$ and $F_{\mathrm{rs2}}(x)$

ported in references [97], [98], [99] and [100] respectively, are directly adopted for comparison in this section, due to the codes of these algorithms are not available.

Comparison between FOLA and GS-SOMA

GS-SOMA adopts a generalization of surrogate-assisted evolutionary frameworks, and uses a variety of different modeling approaches to approximate the complex problem landscape [97]. Three of the benchmark functions (functions F_{rs5}, F_{rs6} and F_{rs8}) given in Appendix A.3 were used for comparison in reference [97]. In this subsection, the comparison between FOLA and GS-SOMA is carried out based on the three benchmark functions. In order to have a fair comparison, N_{femax} and the number of independent runs are set to the same values given in [97], $i.e.$ N_{femax} =8,000, and the number of independent runs is 20. According to [97], the comparison is based on the average fitness value, standard deviation, the median, the best and the worst of the fitness values. Table 2.9 lists the experimental results obtained by FOLA and GS-SOMA. It can be seen that FOLA finds smaller average fitness values than GS-SOMA on benchmark functions F_{rs5} and F_{rs8}. For function F_{rs6}, GS-SOMA find a smaller mean value.

Table 2.9: Comparison between FOLA and GS-SOMA on functions F_{rs5}, F_{rs6} and F_{rs8}

Functions	Algorithms	Mean	Std.	Median	Best	Worst
F_{rs5}	FOLA	-256.3119	22.7237	-268.2107	-279.2096	-208.7800
	GS-SOMA	-126	16	-123	-164	-99.7
F_{rs6}	FOLA	125.6906	3.3076	124.4483	120.2057	132.0854
	GS-SOMA	119.00	3.05	119	114	124
F_{rs8}	FOLA	-115.2299	6.4369	-115.3666	-124.9800	-103.5452
	GS-SOMA	-112	1.05	-113	-123	-111

Comparison between FOLA and OLPSO

Reference [98] proposed algorithms OLPSO-G and OLPSO-L, which use an orthogonal learning strategy to discover useful information. Two of the benchmark functions given in Appendix A.3 ($i.e.$ functions F_{rs1} and F_{rs4}) were used for comparison in reference [98], as well as other two functions, which derive from functions F_{rs3} and F_{rs5} by removing the bias and shift. To distinguish from functions F_{rs3} and F_{rs5}, the two derived functions are called F_{rs3} and F_{rs5} without shift and bias. Since there is no bias in the two derived functions, their $F_{\min}$ changes to 0. According to reference [98], N_{femax} is set to 200,000, the number of independent runs is set to 25 for FOLA, and the comparison is carried out based on the average fitness value and standard deviation. Table 2.10 lists the experimental results obtained by FOLA, OLPSO-G and OLPSO-L. For functions F_{rs1} and F_{rs5} without shift and bias, FOLA outperforms OLPSO-G and OLPSO-L. As for F_{rs4}, FOLA and OLPSO-L are considered to have the same performance, as the gap between their results is less than 10^{-5}. OLPSO-G and OLPSO-L have the same performance in solving function F_{rs3} without shift and bias.

Comparison between FOLA and SOPEN

Reference [99] introduced algorithm SOPEN, which is derived from the idea of vector potential field. According to reference [99], the comparison is based on functions F_{rs1} and F_{rs4}, but without shift and bias; $F_{\min}$ of the functions is 0; the

Table 2.10: Comparison between FOLA and OLPSO on four functions

Functions	Algorithms	Mean	Std.
F_{rs1}	FOLA	390.3189	1.1039
	OLPSO-G	424.75	34.80
	OLPSO-L	415.95	23.96
F_{rs4}	FOLA	-330.00	3.5E-10
	OLPSO-G	-328.575	1.04
	OLPSO-L	-330.00	1.64E-14
F_{rs3} without shift and bias	FOLA	21.2618	0.1674
	OLPSO-G	7.69E-15	1.78E-15
	OLPSO-L	4.28E-15	7.11E-16
F_{rs5} without shift and bias	FOLA	37.6426	6.152
	OLPSO-G	46.09	12.88
	OLPSO-L	53.35	13.35

search range is set to [-2.048,2.047] for F_{rs1} without shift and bias; the search range of F_{rs4} without shift and bias is [-5.12,5.11]; N_{femax} is set to 75,000; the number of independent runs is set to 20; and the comparison is carried out based on mean and minimum of the fitness values. Table 2.11 lists the experimental results obtained by FOLA and SOPEN. FOLA outperforms SOPEN in solving 30- and 100-dimensional function F_{rs1} without shift and bias, while has the same performance as SOPEN on 30- and 100-dimensional function F_4 without shift and bias.

Reference [99] also investigates the number of FEs required to reach a specified stopping criteria when solving 100-dimensional functions. Table 2.12 gives the numbers of FEs required by FOLA and SOPEN to reach the stopping criteria. For 100-dimensional functions F_{rs1} and F_{rs4} without shift and bias, FOLA requires less FEs, in comparison with SOPEN. This suggests that FOLA converges faster than SOPEN in this case. Reference [99] also provides the computation time consumed by SOPEN when solving 100-dimensional functions. However, computation time varies according to the configuration of the computer used. Therefore the compari-

Table 2.11: Comparison between FOLA and SOPEN on 30- and 100-dimensional functions F_{rs1} and F_{rs4} without shift and bias

Functions	Algorithms	Mean	Minimum
30-dimensional F_{rs1} without shift and bias	FOLA	0.3826	0.0127
	SOPEN	23.91	18.57
30-dimensional F_{rs4} without shift and bias	FOLA	1.11E-10	6.9713E-011
	SOPEN	0	0
100-dimensional F_{rs1} without shift and bias	FOLA	90.8251	88.9556
	SOPEN	95.23	92.87
100-dimensional F_{rs4} without shift and bias	FOLA	8.4684E-010	6.0799E-010
	SOPEN	0	0

son of computation time between FOLA and SOPEN is not carried out here.

Table 2.12: Comparison between FOLA and SOPEN on 100-dimensional functions F_{rs1} and F_{rs4} without shift and bias

Functions	Stopping Criterion	Algorithms	FEs required
100-dimensional F_{rs1} without shift and bias	10^2	FOLA	6,835
		SOPEN	36,000
100-dimensional F_{rs4} without shift and bias	10^{-3}	FOLA	15,639
		SOPEN	28,925

Comparison between FOLA and SamACO

Reference [100] introduces algorithm SamACO, which simulates the foraging behavior of a group of ants and focuses on continuous variable sampling. Among the benchmark functions given in Appendix A.3, five functions, functions $F_{\mathrm{rs1}} \sim F_{\mathrm{rs5}}$, were used for comparison in reference [100]. N_{femax} is set to 300,000. The mean and standard deviation of error values (the distance between the obtained fitness value and F_{min}) are concerned here. Table 2.13 lists the experimental results obtained by FOLA and SamACO. It can be seen that FOLA outperforms SamACO in solving functions F_{rs1}, F_{rs2} and F_{rs5}. FOLA and SamACO have the same performance for function F_{rs4}. As for function F_{rs3}, SamACO obtains the mean value of 20, while FOLA has the value of 20.8754.

Table 2.13: Comparison between FOLA and SamACO on five benchmark functions

Functions	Algorithms	Mean	Std.
F_{rs1}	FOLA	0.1329	0.7279
	SamACO	126	294
F_{rs2}	FOLA	0.010	0.0161
	SamACO	0.0167	0.0146
F_{rs3}	FOLA	20.8754	0.0928
	SamACO	20.0	4.3E-3
F_{rs4}	FOLA	1.06E-12	9.6E-13
	SamACO	1.59E-14	2.6E-14
F_{rs5}	FOLA	43.4796	9.2606
	SamACO	270	86.9

2.5 Conclusions

FOLA capitalises on the merits of the structure of multiple automata, the dimensional search, the dividing of the dimensional search domain into cells, and the memories of the performance evaluation of the dimensional states in a form of cell values. By these approaches, FOLA is able to undertake search in continuous states and achieve accurate solutions efficiently. There are two key parameters, w_{c} and N_{femax}, to tune in FOLA when it is applied to resolve a specific application problem. The two parameters can be determined with the basic knowledge of the range of variables involved and the solution accuracy required in the application problem.

The simulation studies have been undertaken on 13 widely used 30-dimensional benchmark functions which include uni-modal and multi-modal problems. The experimental results have shown that FOLA is able to achieve more accurate results than the other eight EAs in finding a global minimum solution, and is more reliable, for its standard deviation of the results over different independent runs is much smaller than that of other algorithms. To investigate whether FOLA can be scaled up to handle the optimisation problems which are highly dimensional, FOLA has been applied to solve the 300-dimensional multi-modal functions which are extremely difficult to solve due to a large number of local minima. In comparison with the other four EAs, FOLA presents its great superiority, as it finds much more accurate solutions, and significantly improves the efficiency and convergence rate.

In addition, FOLA has also been applied to solve nine challenging multi-modal benchmark functions, which are rotated and shifted. In this case, FOLA is compared with two popularly used EAs and four newly-proposed EAs. The experimental results have shown that FOLA offers better performance for most of the benchmark functions, in terms of the accuracy of the obtained optimal solutions and the convergence rate. FOLA is able to reduce the computation time greatly, especially for high-dimensional functions.

Chapter 3

Multi-objective Optimisation by Learning Automata

3.1 Introduction

Many (perhaps most) real-world problems are, in fact, multi-objective optimisation problems. Unlike single-objective optimisation, whose goal is to find the global maximum or minimum subject to an objective function, a multi-objective optimisation problem has usually no unique, perfect solution, but a series of non-inferior alternative solutions, known as Pareto optimal solutions, which represent the possible trade-off among conflicting objectives. The multi-objective optimisation problems can be formulated as follows:

$$\text{Minimise } \ F(X) = [f_1(X), f_2(X), \cdots, f_{m_\mathrm{f}}(X)] \tag{3.1.1}$$

$$\text{s.t. } \ g_i(x) \leq 0, i = 1, 2, \cdots, m_\mathrm{g}$$

where $X = [x_1, x_2, \cdots, x_N]^T \in \mathbb{R}^N$ is the vector of variables to be optimised, functions g_i $(i = 1, 2, \cdots, m_\mathrm{g})$ are constraint functions of the problem, and functions f_i $(i = 1, 2, \cdots, m_\mathrm{f})$ are m_f objective functions. m_f fitness values, obtained by applying solution X in the m_f objective functions, compose an objective function vector.

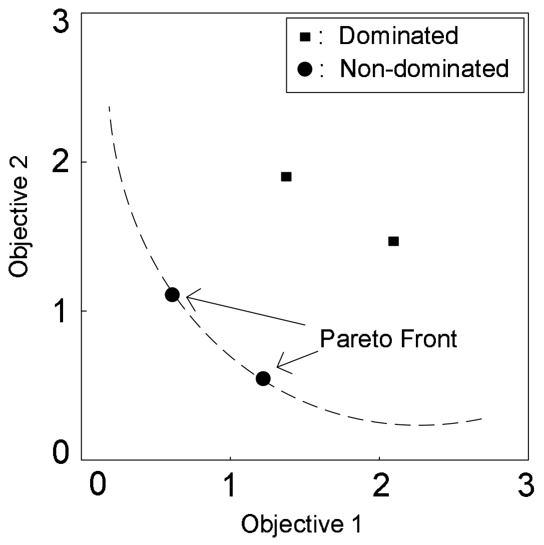


Figure 3.1: Dominance relation in multi-objective problems

The optimal solutions of a multi-objective optimisation problem can be described by the concept of Pareto dominance and Pareto optimality, which are mathematically defined as follows [101][102]:

Definition 1 (Pareto Dominance): A vector $U = [u_1, u_2, \cdots, u_{m_\mathrm{f}}]$ *is said to dominate* $V = [v_1, v_2, \cdots, v_{m_\mathrm{f}}]$ *if and only if* U *is partially less than* V*, i.e.* $\forall i \in \{1, 2, \cdots, m_\mathrm{f}\}, u_i \leq v_i \ \wedge \ \exists i \in \{1, 2, \cdots, m_\mathrm{f}\} : u_i < v_i$.

Definition 2 (Pareto Optimality): A solution X_U *is said to be Pareto optimal if and only if there is no* X_V *for which* $V = F(X_V) = (v_1, v_2, \cdots, v_{m_\mathrm{f}})$ *dominates* $U = F(X_U) = (u_1, u_2, \cdots, u_{m_\mathrm{f}})$.

As an example to explain this concept, assume that a multi-objective problem is to minimise $F(X_a), a \in \{U, V\}$, where $U = F(X_U) = (3.25, 1.76, 4.67)$ and $V = F(X_V) = (3.15, 1.76, 4.22)$. In this example, objective function vector U is dominated by V, and V is non-dominated in the objective space. X_V is the Pareto optimal solution (also called non-dominated solution) of set $\{X_U, X_V\}$, and V is Pareto optimal objective function vector (also called non-dominated objective function vector). A set of all the Pareto optimal solutions is called the Pareto set, which can be used to form a Pareto front in the objective space, as shown in Fig. 3.1.

It can be seen that the aim of multi-objective optimisation is to gain the Pareto set whose Pareto optimal objective function vectors are evenly distributed on the Pareto front [2]. The Pareto front is evaluated through two aspects: 1) convergence to the Pareto-optimal set and 2) maintenance of diversity among the solutions of the Pareto-optimal set.

In order to obtain an accurate Pareto front, there are two standard methods for treating multi-objective problems. One is to convert multi-objective problems into a single objective problem using the weighted-sum method or weighted Tchebycheff method; and the other one is Pareto front-based method, which applies a population of individuals, and each of them represents one Pareto optimal solution. Based on this concept, various algorithms have been proposed to solve multi-objective optimisation problems in the past few decades, such as multi-objective evolutionary algorithms [103][104], multi-objective genetic algorithms [29][7] and group search optimiser [60]. These multi-objective algorithms propose mathematical improvements to meet the demand of solving problems. They have been comprehensively investigated in various application areas, such as power plant [43], wireless sensor networks [30], structural mechanics problems [105], and other engineering problem [31], and so on. The first two applications adopt the weighted-sum methods, which are only capable of solving convex Pareto front problems but have a difficulty in solving the multi-objective problems whose Pareto fronts are non-convex [7]. Nonetheless, there is no way to predetermine if a problem is convex or concave in many applications. The latter is an NSGA II-based method, which suffers from the drawback of high computational complexity caused by non-dominated sorting.

This chapter presents a novel method for multi-objective optimisation by learning automata (MOLA). MOLA adopts the strategy of multiple automata, dimensional search and action selection, which are similar to those in FOLA. Unlike FOLA, the reinforcement signal adopted here is generated through comparing the state with all the non-dominated solutions found so far. MOLA mainly comprises two processes: the process of searching and the process of learning from neighborhood. The process of searching is carried out through a tournament that is held between Pareto global search and Pareto local search. This tournament can lead to a better trade-off between exploitation and exploration, which is a critical factor in finding the optimal solution. In the process of learning, the relationship of neighborhood among the non-dominated solutions is investigated, as it is believed that useful information that can benefit the search is embeded in neighborhood. Based on the relationship, non-dominated solutions are updated based on their neighbors, in order to fully explore the probability of finding the Pareto set. In the search process, an elite list is used to keep track of the non-dominated solutions found. Once a new non-dominated solution is found, it will be included in the elite list. Simultaneously, the elite list will dispose the solutions that were previously stored in the list but are no longer qualified, as they are dominated by the newly-found non-dominated solution. At the end of the optimisation process, the non-dominated solutions stored in the elite list form a Pareto front, which is the aim of the multi-objective optimisation. MOLA has been found being capable and efficient in finding accurate solutions of complex optimisation problems. The merits of MOLA have been demonstrated, in comparison with both weighted-sum methods and Pareto front-based methods. The study is undertaken on a number of complex benchmark functions, which include a wide range of multi-objective models. The simulation results have shown that MOLA is superior over the other algorithms with respect to the accuracy of the non-dominated solutions and the spreadout of the Pareto front.

3.2 The MOLA Method

MOLA consists of N automata, and each automaton is responsible for searching on a specified dimension. Similar to FOLA, the ith learning automaton of MOLA can be defined as $\langle \chi_i, A_i, r, P_i, \mathcal{T} \rangle$, in which χ_i, A_i, r, P_i and $\mathcal{T}$ denote the set of dimensional state x_i, possible actions, reinforcement signal, actions probabilities and reinforcement scheme respectively. Besides, each dimension is divided into D cells, and the cell value of cell $c_{i,j}$ is denoted as $V(c_{i,j})|_{x_i \in c_{i,j}}$.

In MOLA, $F(X)$ is an N-dimensional multi-objective minimisation (or maximisation) problem to be resolved subject to constraints applied to $F(\cdot)$ or/and X, where $F(\cdot)$ is composed of multiple objective functions, $i.e.$ $F(X) = [f_1(X), \cdots, f_{m_f}(X)]$, which will be used by the environment to generate reinforcement signal r, and X

is a solution in the N-dimensional space, denoted by $X = [x_1, \cdots, x_i, \cdots, x_N]$, where $x_i \in \chi_i$. In MOLA, Pareto set obtained during the search process is stored in an elite list, denoted as $\mathbb{L}$, to keep track of all the non-dominated solutions found so far. The updating rule of $\mathbb{L}$ will be given in Section 3.2.2.

3.2.1 An automaton and its reinforcement scheme

MOLA adopts the concepts of path value, cell value and path selection, which are the same as those employed in FOLA (refer to Section 2.2.1). To better explain MOLA, the three concepts are introduced briefly as follows:

- A path value is to estimate the potential of finding a better solution if MOLA searches down on the path from a dimensional state. Two estimated path values, denoted by $L_l(x_i)$, $l = 1, 2$, are to be found with respect to dimensional state x_i. The value of a path can be estimated as follows:

$$L_l(x_i) = (1 - \lambda_1) \sum_{m=1}^{k-1} \lambda_1^{m-1} v_m^* + \lambda_1^{k-1} v_k^* \qquad l = 1, 2 \tag{3.2.1}$$

  where $v_{l,m}^*$ denotes the mth element of the vector in which the k cell values, locating on path l, are reordered in descending order.

- Action selection employs two probabilities, p_1 and p_2. The former is used to select a path which the current dimensional state moves to, while the latter is applied to select the cell on the selected path. With the availability of the path values, the left path L_1 or the right path L_2 is chosen with a probability given as follows:

$$p_1(L_l(x_i)) = \frac{e^{L_l(x_i)/\tau}}{\sum_{s=1}^{2} e^{L_s(x_i)/\tau}} \qquad l = 1, 2 \tag{3.2.2}$$

  Suppose the right path is selected. Then a cell, which the current dimensional state moves to, is selected, among the k cells located on the path, according to the probability that is calculated from the cell values of the k cells as follows:

$$p_2(V(c_{i,j+s})|_{x_i \in c_{i,j+s}}) = \frac{e^{V(c_{i,j+s})|_{x_i \in c_{i,j+s}}/(2\tau)}}{\sum_{z=1}^{k} e^{V(c_{i,j+z})|_{x_i \in c_{i,j+z}}/(2\tau)}} \tag{3.2.3}$$

  where $s = 1, 2 \cdots, k$.

- As one of the major functions of memory, updating cell values relies on the reinforcement signal of current dimensional state x_i, $i.e.$ $r(X(x_i))$, and the weighted path values of possible paths, $i.e.$ the two path values, $L_1(x_i)$ and $L_2(x_i)$, obtained according to (3.2.1). The cell value of cell $c_{i,j}$, which the current dimensional state x_i locates in, is updated as follows:

$$\begin{aligned} V(c_{i,j})|_{x_i \in c_{i,j}} \leftarrow r(X(x_i)) + \alpha V(c_{i,j})|_{x_i \in c_{i,j}} + \\ (1-\alpha)\big((1-\lambda_2)L_{\max}(x_i) + \lambda_2 L_{\min}(x_i)\big) \end{aligned} \tag{3.2.4}$$

where $L_{\max}(x_i)$ and $L_{\min}(x_i)$ are the two estimated path values at dimensional state x_i, while satisfying $L_{\max}(x_i) = \max\{L_1(x_i), L_2(x_i)\}$ and $L_{\min}(x_i) = \min\{L_1(x_i), L_2(x_i)\}$.

Reinforcement signal

Similar to FOLA, a reinforcement signal of the search action should be provided to evaluate the effectiveness of the selected action. However, the design of reinforcement signal in MOLA is different from that in FOLA, due to the different nature of objective setting. Assume the dimensional state moves from x_i to x_i' after an action is taken. The fitness values of state $X(x_i')$ can be obtained by applying $X(x_i')$ in the objective functions of the minimisation problem, $F(\cdot)$. Suppose dimensional state x_i' locates in cell $c_{i,j}$. Then a reinforcement signal is assigned to cell $c_{i,j}$ by comparing $X(x_i')$ with the non-dominated solutions found previously, according to the following rule:

$$r(X(x_i')) = \begin{cases} 1 & \text{if } X(x_i') \text{ is non-dominated} \\ 0 & \text{otherwise} \end{cases} \tag{3.2.5}$$

3.2.2 Forming the Pareto set

All the non-dominated solutions found in the optimisation process are stored in the elite list $\mathbb{L}$. Suppose the state moves from $X(x_i)$ to $X(x_i')$. If the current state, $X(x_i')$, is not dominated by the solutions that have been stored in $\mathbb{L}$ previously, it will be denoted as X_{best}. The relationship between state X and X_{best} can be described by the following rule:

$$X_{\text{best}} \leftarrow \begin{cases} X(x'_i) & \text{if } X(x'_i) \text{ is non-dominated} \\ X_{\text{best}} & \text{otherwise} \end{cases} \tag{3.2.6}$$

where $X(x'_i) = [x_1, \ldots, x_{i-1}, x'_i, x_{i+1}, \ldots, x_N]$. Then $\mathbb{L}$ is updated based on the following rule:

$$\mathbb{L} \leftarrow \begin{cases} \mathbb{L} \cup \{X_{\text{best}}\} - B & \text{if } r = 1 \\ \mathbb{L} & \text{otherwise} \end{cases} \tag{3.2.7}$$

where B is a set of the solutions which were stored in the elite list previously but currently are dominated by X_{best}, as formulated in the following equation:

$$B = \{X : X \in \mathbb{L}, F(X) \succeq F(X_{\text{best}})\} \tag{3.2.8}$$

It can be seen that once a non-dominated solution is found, it will be included in the elite list and labeled as X_{best}. For those solutions which are dominated by X_{best}, they will be unqualified and disposed from the elite list, in order to ensure that only non-dominated solutions are stored in the elite list. At the end of the optimisation process, elite list $\mathbb{L}$ will be used to form a Pareto set.

3.2.3 The process of searching and learning

The calculation of MOLA consists of two parts: the process of searching and the process of learning from neighborhood. They are introduced in the following two subsections respectively.

The process of searching

The process of searching is carried out through a tournament that is held between two types of search, Pareto global search and Pareto local search. The rule of the tournament will be introduced in Section 3.2.4. In this subsection, Pareto global search and Pareto local search are introduced respectively as follows:

Pareto global search (PGS): Starting with the first state X, by selecting automaton i, at dimensional state x_i, a search is undertaken by calculating the path values of two possible paths using the cell values obtained in the previous iterations, as given in (3.2.1), and then moving x_i towards a path, which is selected according to the probability calculated using the path values, as in (3.2.2), with a step length η, which is determined by the probability calculated from the cell values of the cells locating on the path, as in (3.2.3). A reinforcement signal is generated using dimensional state x_i, according to (3.2.5). Then the estimated path values and the reinforcement signal are used to update the cell value of $c_{i,j}$, in which dimensional state x_i locates, using (3.2.4). In this case the objective function values of the multiple objective functions are calculated once, and the cell value of $V(c_{i,j})|_{x_i \in c_{i,j}}$ is updated to replace the cell value that was stored previously when any dimensional state located within cell $c_{i,j}$ was visited. At the same time, X_{best} and elite list $\mathbb{L}$ are updated according to (3.2.6) and (3.2.7) respectively. If the reinforcement signal is not equal to 1, the current iteration ends and the next automaton is selected in order for MOLA computation to continue. Otherwise searching on this path is regarded worthy, as the cells locating on this path could be very sensitive and need more exploitation. In order to further exploit the potential of the dimensional states located on this path, the search action is taken continuously for I_{emax} times in the same way presented in (3.2.1), (3.2.2), (3.2.3), (3.2.5), (3.2.4), (3.2.6) and (3.2.7). An iteration completes after the exploitation, which has updated X_{best} and all values of the cells visited, before the next automaton is selected. After all automata are sequentially selected once, in order to increase the diversity and fully explore the probability of finding the Pareto set, different perturbations are introduced to the non-dominated solutions stored in the elite list, according to the following rule:

$$X' \leftarrow X + \Delta + \beta(X_{\text{best}} - X) \tag{3.2.9}$$

where $\Delta_i = \text{sign}(\kappa)\zeta(x_{\text{max},i} - x_{\text{min},i})$, ζ is a random variable and $\zeta \in [0, \frac{k}{D}]$. The sign function is used to choose the moving direction of x_i. Suppose x_i is located in cell $c_{i,j}$, then the input to the sign function is the subtraction of the two adjacent cell values of $c_{i,j}$, that is, $\kappa = (V(c_{i,j+1}) - V(c_{i,j-1}))$. If $\kappa \geq 0$, $\text{sign}(\kappa) = 1$, otherwise, $\text{sign}(\kappa) = -1$. It can be seen that x_i moves towards the direction in which the adjacent cell has a larger cell value. Once completing the perturbation operation, the obtained solutions are used to update X_{best} and $\mathbb{L}$, according to (3.2.6) and (3.2.7)

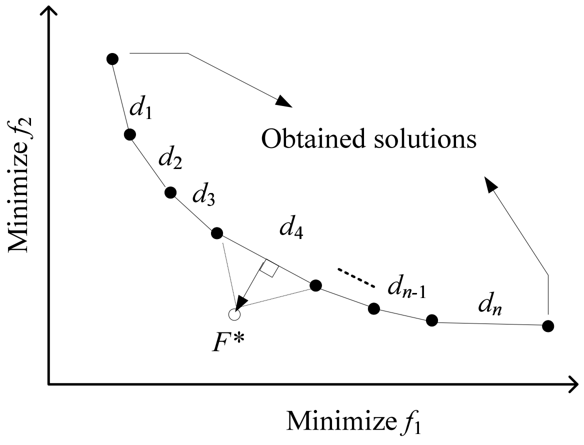


Figure 3.2: The illustration of finding F^*

respectively.

Pareto local search (PLS): PLS is similar to PGS with the exception of the following two aspects:

1. The perturbation operation used in PGS, $i.e.$ (3.2.9), is not performed in PLS.

2. The calculation of reinforcement signal is different from that used in PGS. To obtain the reinforcement signal, a target objective function vector, F^*, which is regarded as the aim of the PLS, should be set at the beginning of the search. F^* can be determined by the following method. First, the Euclidean distance between every two consecutive objective function vectors is calculated, as given in Fig. 3.2, in which solid circles denote the objective function vectors of the solutions stored in $\mathbb{L}$. The pair of the two consecutive objective function vectors, which has the largest Euclidean distance, is taken as two base vertexes to create an isoceles triangle, in which the height is half of the base length. Then the third vertex of the triangle is selected as F^*, as given in Fig. 3.2.

   During the process of Pareto local search, the fitness value of state $X(x'_i)$ can be obtained by calculating the Euclidean distance between F^* and the objective function vector of the solution, $i.e.$ $F(X(x'_i))$. For simplicity, the Euclidean distance between F^* and $F(X(x'_i))$ is denoted as $d(F^*, F(X(x'_i)))$.

Then a reinforcement signal is generated by the following rule:

$$r(X(x_i')) = \begin{cases} 1 & \text{if } d(F^*, F(X(x_i'))) \leq d(F^*, F(X_{\text{lbest}})) \\ 0 & \text{otherwise} \end{cases} \tag{3.2.10}$$

where X_{lbest} (called the local best solution) denotes the latest best solution found during the process of Pareto local search. In other words, among all the solutions found during the process of Pareto local search, the objective function vector of solution X_{lbest} is closest to F^*. It can be seen that $r = 1$ represents a favorable response, as the objective function vector of the obtained solution $X(x_i')$ is closer to F^* than that of X_{lbest}; $r = 0$ is an unfavorable response.

The relationship between state $X(x_i')$ and X_{lbest} can be described by the following equation:

$$X_{\text{lbest}} \leftarrow \begin{cases} X(x_i') & \text{if } r = 1 \\ X_{\text{lbest}} & \text{otherwise} \end{cases} \tag{3.2.11}$$

where $X(x_i') = [x_1, \ldots, x_{i-1}, x_i', x_{i+1}, \ldots, x_N]$. In the process of Pareto local search, X_{lbest} is updated according to (3.2.11), and at the same time, X_{lbest} is used to update X_{best} and $\mathbb{L}$ according to (3.2.6) and (3.2.7) respectively.

The process of learning

Suppose the current state, X, is one of the non-dominated solutions stored in elite list $\mathbb{L}$, as shown in Fig. 3.3, which plots the objective function vectors, $F(\cdot)$, of the non-dominated solutions in the objective space. The solutions whose objective function vectors locate at the adjacent area of $F(X)$, $e.g.$ area E_1, are the neighboring solutions of X. It is believed that the neighboring solutions of X carry more useful information that can benefit the search around X, and they can make more contributions than remote solutions, $e.g.$ the solutions whose $F(\cdot)$ locates at area E_2.

Here, a method of clarifying the relationship of neighborhood among the non-dominated solutions is given first. Then the operation of learning from neighborhood can be performed based on the obtained solutions.

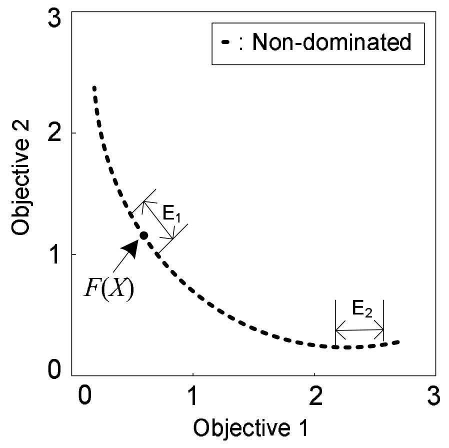


Figure 3.3: The illustration of X's neighborhood

- To find out the relationship among the non-dominated solutions, an uniform spread of M weight vectors, ($\mathbf{W}^1, \cdots, \mathbf{W}^i, \cdots, \mathbf{W}^M$), where $\mathbf{W}^i = [w_1, \cdots, w_{\text{mf}}]$ and $\sum(\mathbf{W}) = 1$, are initialized at the beginning of the learning process [103]. Then the Euclidean distances between any two weight vectors are calculated, and the $M/4$ closest weight vectors to each weight vector are found according to the Euclidean distances. Let the indexes of the $M/4$ closest weight vectors to $\mathbf{W}^i$ be denoted as set $D(i)$.

  Each weight vector, $\mathbf{W}^i$ ($i = 1, 2, \cdots, M$), is associated with a subsolution, denoted as X^i_{sub}, which is selected from $\mathbb{L}$ according to Tchebycheff rule as follows:

$$X^i_{\text{sub}} = \text{argmin}_{X\in\mathbb{L}} g^{\text{te}}(X|\mathbf{W}^i, z^*) \tag{3.2.12}$$

  where Tchebycheff value is defined as:

$$g^{\text{te}}(X|\mathbf{W}^i, z^*) = \max_{1\le j\le \text{mf}} \{w_j \frac{\mid f_j(X) - z^*_j \mid}{f_{j,\text{range}}}\} \tag{3.2.13}$$

  and $z^* = (z^*_1, \cdots, z^*_j, \cdots, z^*_{\text{mf}})$ is the reference point, $i.e.$ $z^*_j = \min\{f_j(X) \mid X \in \mathbb{L}\}$; $f_{j,\text{range}}$ is the estimated range of f_j, $i.e.$ $f_{j,\text{range}} = \max\{f_j(Y) \mid Y \in \mathbb{L}\} - \min\{f_j(X) \mid X \in \mathbb{L}\}$.

  After each weight vector is assigned with a subsolution, the neighboring solutions of X^i_{sub} comprise the subsolutions of the weight vectors whose indexes belong to set $D(i)$. The set of the neighboring solutions of X^i_{sub} is denoted as set $\text{Nei}(X^i_{\text{sub}})$, and it can be formulated as:

$$\text{Nei}(X^i_{\text{sub}}) = \{X^j_{\text{sub}} \mid j \in D(i)\} \tag{3.2.14}$$

- With the neighborhood classification introduced above, each subsolution X^i_{sub} is updated based on its neighboring solutions. Two indexes will be randomly selected from $D(i)$. Suppose the two indexes are denoted as $r1$ and $r2$. Then a new solution can be generated according to the following learning operation:

  $$\bar{y} = X^i_{\text{sub}} + 0.5 \times (X^{r1}_{\text{sub}} - X^{r2}_{\text{sub}}) \tag{3.2.15}$$

  $\bar{y}$ will be used to replace X^i_{sub} if its Tchebycheff value is smaller than that of X^i_{sub}. Simultaneously, $\bar{y}$ will be used to update X_{best} and $\mathbb{L}$ according to (3.2.6) and (3.2.7) respectively if it is not dominated by any solution stored in $\mathbb{L}$.

3.2.4 The implementation of MOLA

The computation of MOLA consists of a certain number of episodes, and each episode includes the process of searching (Section 3.2.3) and the process of learning from neighborhood (Section 3.2.3). Within one episode, the search method can adopt either PGS or PLS, or both of them, depending on a tournament between PGS and PLS. In the tournament, the search method which finds more new non-dominated solutions is regarded as the winner. The operation of selecting the winner and the subsequent operations performed by the winner compose one cycle of the tournament, which requires four episodes, as illustrated in Fig. 3.4. It can be seen that both PGS and PLS are performed in the first episode, and a winner is selected between PGS and PLS according to the number of new non-dominated solutions they have found. For the following three episodes, only the winner is performed in the process of searching. Then, another cycle of the tournament is performed for the next four episodes. This cycle continues until the following condition is met: if one of the two search methods can not find a single new non-dominated solution continuously for five episodes, it will be withdrawn and will not be performed in

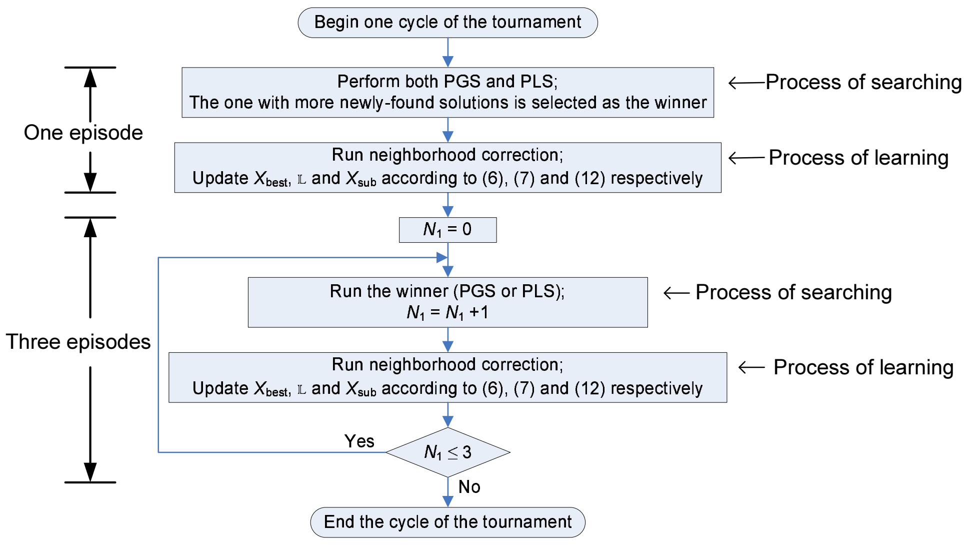

Begin one cycle of the tournament
Perform both PGS and PLS; The one with more newly-found solutions is selected as the winner
Process of searching
One episode
Run neighborhood correction; Update X_{best}, $\mathbb{L}$ and X_{sub} according to (6), (7) and (12) respectively
Process of learning
$N_1 = 0$
Run the winner (PGS or PLS); $N_1 = N_1 + 1$
Process of searching
Three episodes
Run neighborhood correction; Update X_{best}, $\mathbb{L}$ and X_{sub} according to (6), (7) and (12) respectively
Process of learning
Yes
$N_1 \leq 3$
No
End the cycle of the tournament

Figure 3.4: The flowchart of one cycle of the tournament

the rest of the optimisation process. It means that for the following episodes, only the other search method and the learning process are performed in one episode. The MOLA computation proceeds in episodes, until a given maximum number of objective function evaluations, N_{femax}, is reached.

3.3 Compared with Weighted-sum Based Algorithms

As mentioned in Section 3.1, there are mainly two classes of approaches to resolve multi-objective problems. In this section, MOLA is compared with the first class of approaches, $i.e.$ the weighted-sum based methods. MOLA is fully compared with two popular weighted-sum based algorithms: Multi-Objective Genetic Algorithm (MOGA) [7] and Multi-Objective Particle Swarm optimizer (MOPSO) [106].

3.3.1 Benchmark functions

Comparative simulation studies have been undertaken based on four benchmark functions, which comprise low- and high- dimensional models, convex and non-convex models, and continuous and discontinuous models respectively, as given in Appendix A.4. The parameters setting of MOGA follows [7], which regulates the multi-objective problem to a single-objective one by calculating the weighted average of the fitness functions. Taking a two-objective problem for example, the optimisation is to find the most suitable X, so that it obtains the minimum of $f(X)$, which is defined as follows:

$$f(X) = \psi_1 f_1(X) + \psi_2 f_2(X);\;\; 0 < \psi_1, \psi_2 < 1;\;\; \psi_1 + \psi_2 = 1 \tag{3.3.1}$$

where the step length of weights ψ_1 and ψ_2 is set to 0.05. For the parameters of MOPSO, they are set as recommended in [107]. MOPSO also adopted the method of converting the objectives into one single objective function, as given in (3.3.1). The population size is set to 50 for MOGA and MOPSO. The parameters adopted by MOLA are the same as those in FOLA with the exception of w_c and N_femax, which are set by trial and error.

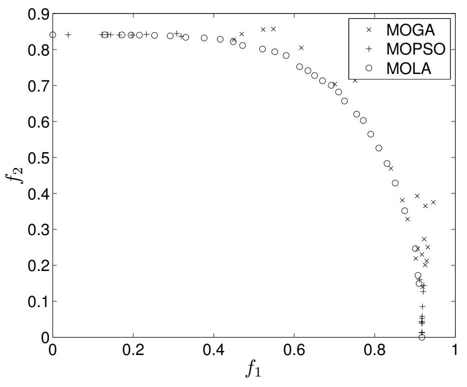
3.3.2 Simulation results

Function I

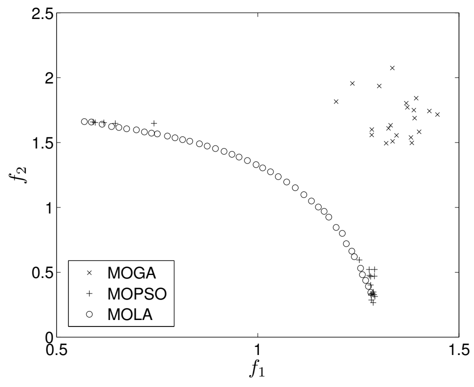
For this type of problem [27], the optimisation algorithms have been applied in a low-dimensional ($N = 2$) and a high-dimensional benchmark function ($N = 30$) respectively. In MOLA, parameter w_c is set to 1 for both 2-dimensional and 30-dimensional cases.

MOGA and MOPSO find only 21 solutions in these cases, for the step length of the weights is set to 0.05. Thus, ψ_1 and ψ_2 can only take 21 different values. Consequently, only 21 solutions can be obtained. Lessening the step length will certainly lead to more solutions, however, the computation cost will increase accordingly and it will not ameliorate the distribution of the solutions.

When the dimensionality of Function I is two, the non-dominated solutions obtained by MOLA, MOGA and MOPSO are plotted in the objective space as shown in Fig. 3.5(a). In order to distinguish the notations of different algorithms, not all non-dominated solutions obtained by MOLA are plotted in the figures provided in this section, but only representative solutions in the objective space are selected to



(a)

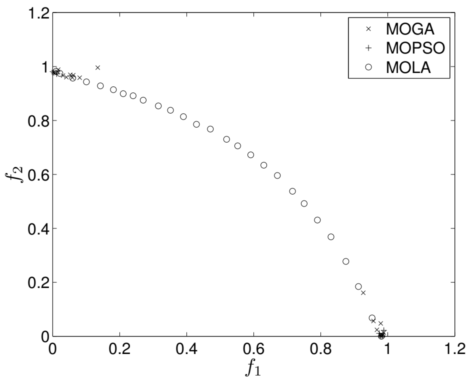


(b)

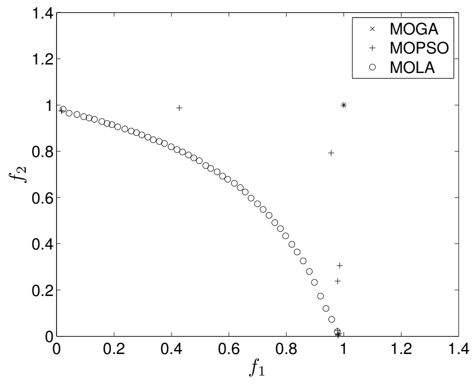
Figure 3.5: Pareto fronts obtained by MOGA, MOPSO and MOLA on Function I: (a) 2 dimensions; (b) 30 dimensions

show the intact shape of Pareto front obtained by MOLA. 3740 non-dominated solutions are found by MOLA with 10,000 FEs. Performing the process 30 times, the total number of solutions obtained by MOLA is not significantly different from each run. The fact that MOLA finds more non-dominated solutions suggests that it can provide more possible solutions that satisfy the optimisation targets. MOLA finds the smooth fronts (that can be considered as the Pareto front) which have the same shape and location in the objective space. For MOGA and MOPSO, 200 iterations ($200 \times 50 = 10,000$ FEs) are performed for each combination of ψ_1 and ψ_2. MOGA can only find solutions at the ends of the Pareto front found by MOLA, due to the fact that the shape of the Pareto front is concave. The solutions of MOPSO gather around the two ends of the Pareto front found by MOLA, however, MOGA does not converge with 200 iterations. If the iterations are sufficient, the solutions obtained by MOGA will also flock to the two ends of the Pareto front obtained by MOLA.

In the case of 30 dimensions, for MOGA and MOPSO, 300 iterations ($300 \times 50 = 15,000$ FEs) are performed for each combination of ψ_1 and ψ_2. Similar to the case of 2 dimensions, the solutions of MOPSO flock to the ends of the front found by MOLA. MOGA cannot converge within 300 iterations, and its solutions begin to flock to the ends of the front when the number of iterations is increased to 1,000. With 15,000 FEs, MOLA finds 665 Pareto non-dominated solutions,



(a)



(b)

Figure 3.6: Pareto fronts obtained by MOGA, MOPSO and MOLA on Function II: (a) 2 dimensions; (b) 30 dimensions

which are much more than those obtained by MOGA and MOPSO. The attraction of MOLA is greatly enhanced by the fact that the solutions found by MOLA can dominate those obtained by MOGA and MOPSO, as shown in Fig. 3.5(b). MOLA presents its superiority over the other two algorithms when the dimensionality of the multi-objective function is large.

Function II

The problem was proposed by Fonseca *et al.* [108]. Both a low dimensionality ($N = 2$) and a high dimensionality ($N = 30$) of the benchmark function have been tested by the optimisation algorithms. In MOLA, parameter w_c is set to 1 for both cases, $i.e.$ 2 dimensions and 30 dimensions.

This problem is a non-convex model which cannot be solved well by MOGA and MOPSO. In the case of 2 dimensions, the solutions obtained by MOGA and MOPSO locate at the two ends of the Pareto front in the objective space, as shown in Fig. 3.6(a). MOLA finds 746 non-dominated solutions with 10,000 FEs.

When the dimensionality of Function II is 30, MOLA performs much better than MOGA and MOPSO, as shown in Fig. 3.6(b). MOGA and MOPSO adopt 300 iterations for each combination of ψ_1 and ψ_2. MOGA can only find one solution (1,1) in the objective space, while MOPSO finds three solutions, including (1,1) and

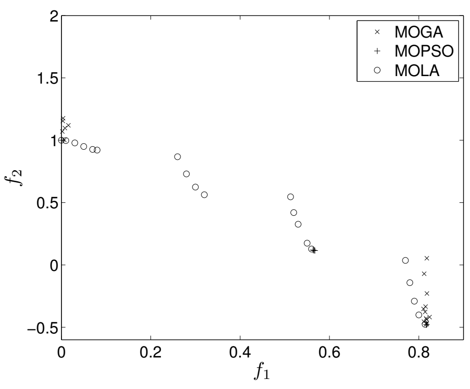


Figure 3.7: Pareto fronts obtained by MOGA, MOPSO and MOLA on Function III

the other two solutions located on the two coordinates respectively. As for MOLA, it finds 401 non-dominated solutions with 15,000 FEs, and these solutions are evenly spread on the Pareto front.

Function III

The problem, which is a discontinuous Pareto front model, was proposed by Deb in 1998 [102]. The true Pareto font for this problem comprises four disconnected lines. In the case, the parameter w_c is set to 0.001 for MOLA.

It can be seen from Fig. 3.7 that results obtained by MOGA and MOPSO with 400 iterations ($400 \times 50 = 20,000$ FEs) are far from satisfactory - only a paucity of solutions have converged to the front, and they are centralized at the ends of the first and fourth disconnected lines. On the second and third disconnected lines, no non-dominated solutions have been found by MOGA and MOPSO. MOLA finds 677 non-dominated solutions with 3,000 FEs, and they are distributed evenly over the four disconnected lines.

Function IV

This function is designed for disc break system, which is proposed by Osyczka *et al.* [109] in 1995. The objectives of the design are to minimise the mass of the brake and the stopping time. The search range of each variable is different, thus,

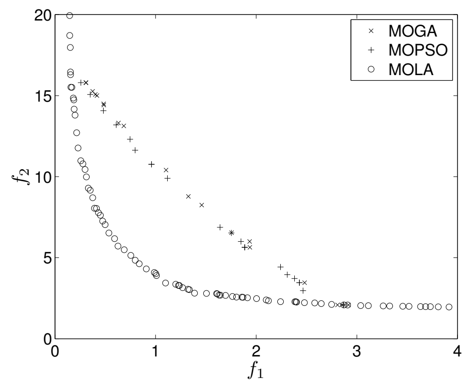


Figure 3.8: Pareto fronts obtained by MOGA, MOPSO and MOLA on Function IV

parameter w_c of MOLA is set to different values for different variables. In this case, each variable is divided into ten grids, thus, $w_{c,i} = [x_{\max,i} - x_{\min,i}]/10$, where $i = 1, \cdots, 4$.

MOPSO and MOGA obtain 21 points respectively with 2,000 iterations. The shape of the Pareto fronts obtained by algorithms MOGA and MOPSO is similar, as shown in the two upper lines in Fig. 3.8. The non-dominated solutions found by MOGA and MOPSO fall sparsely on the curve. This improvement is due to the $F(X)$ is convex. However, in practical applications, usually it is not known in advance whether the target fitness function is convex or not, which is the problem confronted by algorithms MOGA and MOPSO. With 3,660 FEs, MOLA obtains 216 non-dominated solutions, which spread evenly on the Pareto front, *i.e.* the bottom curve in Fig. 3.8. It can be seen that the range of the Pareto front found by MOLA is wider than that obtained by MOGA and MOPSO, and MOLA can find more accurate results than other two algorithms.

3.3.3 Remarking

The computation time

In order to investigate the computation time of MOLA in comparison with MOGA and MOPSO, they are used to optimise Function I in 10, 20, 30, 40 and 50 dimensions respectively. The step length of weights ψ_1 and ψ_2 is set to 0.01 for MOPSO

Table 3.1: The setting of N_{femin} for solving Function I with different dimensionality

N	10	20	30	40	50
N_{femin}	5,000	10,000	15,000	20,000	25,000

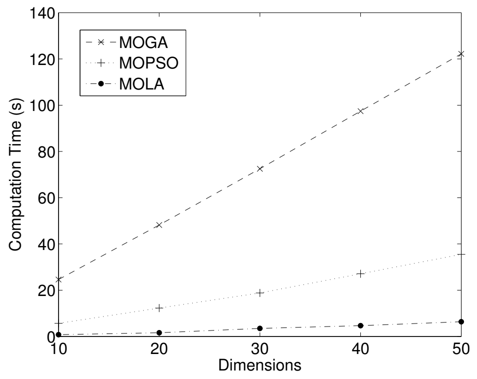


Figure 3.9: The computation time consumed by MOGA, MOPSO and MOLA in solving multi-objective Function I with different dimensions.

and MOGA, which means that they can find 100 non-dominated solutions. Parameter N_{femax} applied by MOLA is listed in Table 3.1; as for MOGA and MOPSO, parameter N_{femax} used for each combination of ψ_1 and ψ_2 is the same as that used in MOLA. The computation time consumed by the three algorithms in solving Functions I is given in Fig.3.9, which shows that the MOLA reduces the computation time in comparison with the other two algorithms. However, when the dimensionality reaches 50, the performance of MOGA and MOPSO is far from satisfactory. As shown in Fig.3.10, the solutions found by MOPSO flock to the two ends; MOGA cannot converge within 25,000 FEs, while MOLA can still find a smooth Pareto front when the dimensionality is large.

The parametric analysis of MOLA

Parameter w_c is related to the resolution of dividing the search space to grids and parameter N_{femax} is used to terminate the computation. Both are concerned with the accuracy of solution. Parameter w_c should be set appropriately to ensure the accuracy of Pareto front obtained by MOLA. With smaller w_c, more accurate Pareto optimal can be obtained. However, a small w_c requires a larger N_{femax} for

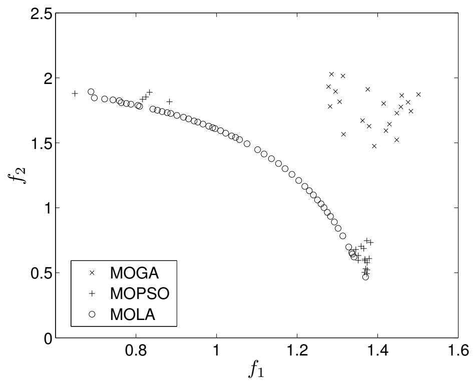


Figure 3.10: Pareto fronts obtained by MOGA, MOPSO and MOLA on 50-dimensional Function I

Table 3.2: The number of non-dominated solutions obtained by MOLA when w_c and N_{femax} are set to different values

w_c	1	0.1	0.01
N_{femax}	15,000	25,000	40,000
Number	525	1601	2391

MOLA to converge. Therefore, w_c is the parameter to tradeoff between the quality of Pareto front and computation load. When both parameters are set to different values, the number of non-dominated solutions obtained by MOLA varies. A large number of non-dominated solutions means it can provide more possible solutions that satisfy the optimisation targets. Table 3.2 lists the number of non-dominated solutions obtained using a range of different w_c and N_{femax}, which are consistent with the analysis. The Pareto fronts obtained when w_c is set to 1 ($N_{\text{femax}} = 15,000$) and 0.01 ($N_{\text{femax}} = 40,000$) respectively are shown in Fig. 3.11. The Pareto front that is obtained when $w_c = 0.01$ is wider in comparison with the case when $w_c = 1$. Both N_{femax} and w_c should be set properly in a specific application so as to ensure the quality of solution and the efficiency of computation.

Convexity of Pareto fronts and weighted-sum methods

The Pareto fronts of multi-objective problems could be convex or non-convex. To solve the problems which have convex Pareto fronts, as illustrated in Fig. 3.1, the

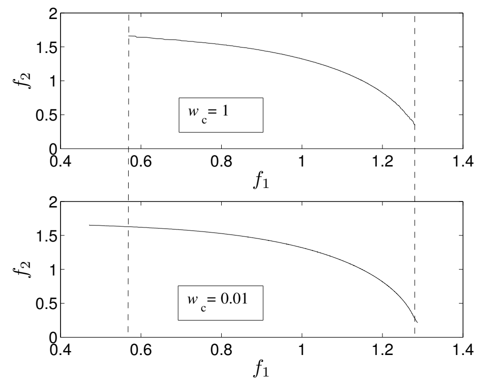


Figure 3.11: Pareto front obtained by MOLA on Function I when w_{c} and N_{femax} are set to different values

weighted-sum methods can be employed to find the whole Pareto front, by specifying different scalar weights to the multiple objectives and then combining multiple objectives into a single composed function, which can be solved by many single-objective optimisation algorithms. However, the solutions obtained by these methods highly depend on the values (more precisely, the relative values) of the weights specified. For example, as shown in Fig. 3.12, the purpose is to find the Pareto front of the problem whose objectives are f_1 and f_2. The solution, obtained by employing weights [0.5 0.5] ($i.e.$ to minimise $0.5 \times f_1 + 0.5 \times f_2$), is denoted by solid circle, and that found by minimising $0.3 \times f_1 + 0.7 \times f_2$ is denoted by solid square. It can be seen that in order to find the whole Pareto front, the number of weight combinations should be large enough. It may be noticed that weighted-sum methods are essentially subjective, due to that the weights which represent one's favor among multiple objectives are required.

3.4 Compared with Pareto Front-based Algorithms

3.4.1 Performance metrics

If the Pareto fronts found by two algorithms are quite different, the performance between the algorithms can be evaluated by directly observing the Pareto fronts. However, sometimes, the Pareto fronts found by different algorithms are similar vi-

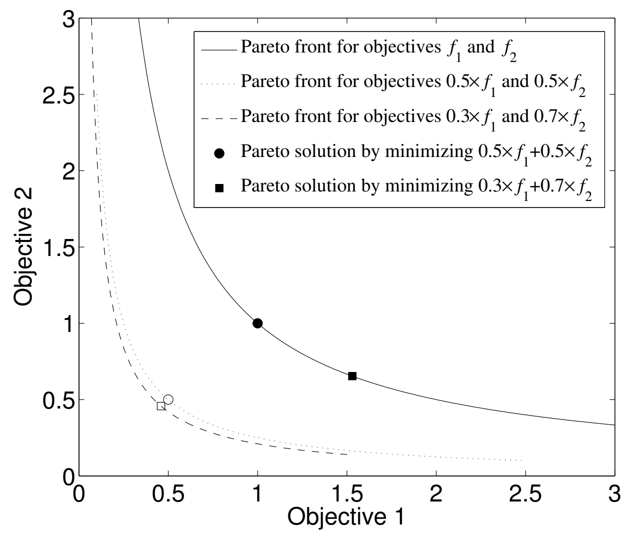


Figure 3.12: Illustration of weighted-sum methods

sually, and it is difficult to distinguish between them with bare eyes. In this case, performance metrics are required to assist in the performance evaluation of the algorithms. Four widely used performance metrics are introduced here.

Distance metric

Distance metric, denoted as $\tilde{D}$, measures the extent of convergence to a known set of Pareto-optimal solutions [29]. First, a set of uniformly spaced Pareto-optimal solutions are selected from the true Pareto front in the objective space. For each solution, the Euclidean distances between the solution and all the Pareto-optimal solutions selected on the true Pareto front are calculated, and the minimum Euclidean distance is obtained for the solution. Then the obtained minimum Euclidean distances of all solutions are averaged and used as the distance metric. The smaller the value of this metric is, the better the convergence toward the Pareto front is.

Hypervolume indicator

The hypervolume indicator has been widely used in evolutionary multi-objective optimisation to evaluate the performance of search algorithms [110]. It computes the volume (in the objective space) covered by the non-dominated solutions of the problems in which all objectives are to be minimised, as given in Fig. 3.13. Mathematically, for each solution $F_i \in \mathbb{L}$ ($\mathbb{L}$ denotes the set of the non-dominated solutions), a hypercube v_i is constructed with a reference point and the solution F_i as the diagonal corners of the hypercube. The reference point can simply be found by constructing a vector of the worst objective function values. Thereafter, the union of all hypercubes is found, and its hypervolume (HV) is calculated according to [111]:

$$HV = \texttt{volumn}(\bigcup_{i=1}^{|\mathbb{L}|} v_i) \tag{3.4.1}$$

Higher value of this performance indicator implies more desirable solutions. One property of this indicator is that it measures both convergence to the true Pareto front and diversity of the obtained solutions.

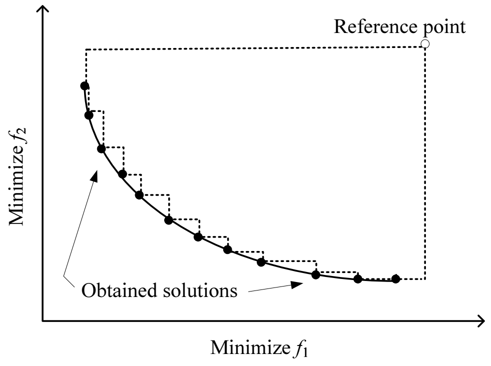


Figure 3.13: The illustration of hypervolume

Diversity metric

Diversity metric $\triangle$ is to measure the extent of spread among the obtained solutions [29]. First, the Euclidean distances between consecutive solutions in the obtained non-dominated set of solutions are calculated and denoted as d_i. Then the average of these distances, $\bar{d}$, can be calculated. Apart from these distances, the calculation of this metric also involves two extreme solutions, which are the two ends of the true Pareto front in the objective space respectively, as given in Fig. 3.14.

Then, the diversity metric can be formulated as:

$$\triangle = \frac{d_f + d_l + \sum_{i=1}^{n-1} |d_i - \bar{d}|}{n+1} \tag{3.4.2}$$

where parameters d_f and d_l are the Euclidean distances between the two extreme solutions and the boundary solutions of the non-dominated set; n is the number of total non-dominated solutions obtained by the algorithm. Smaller value of diversity metric $\triangle$ implies wide and uniform spreadout of the obtained non-dominated solutions.

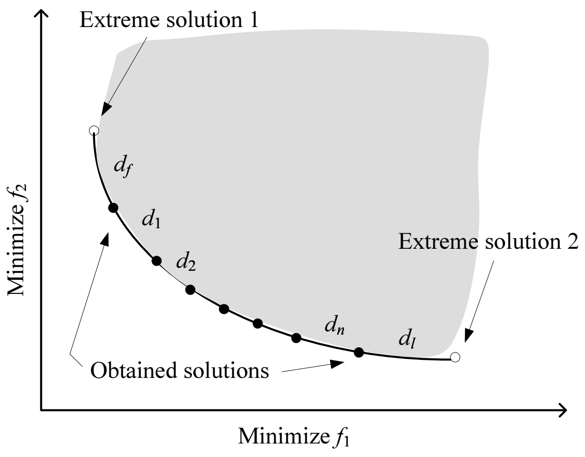


Figure 3.14: The illustration of the diversity metric

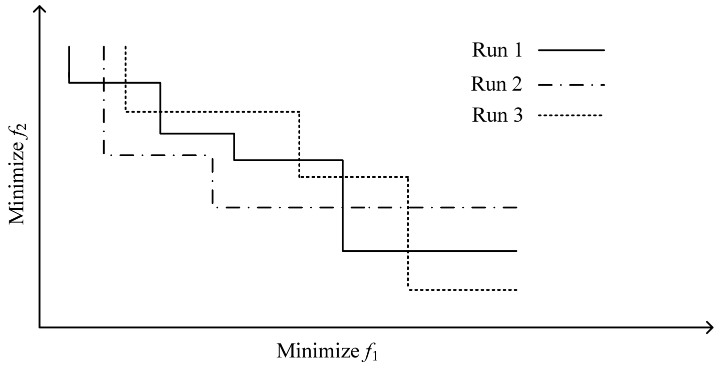
Summary attainment surfaces

Some performance indicators do not adequately express the amount by which one Pareto front should be judged better than another. Looking at the shape of the obtained Pareto front can provide insight into the strengths and weaknesses of an optimiser. Especially when the true Pareto front is known, seeing the distance away from it and coverage along it can provide a supplement to any performance metric. In this performance measure, the outcome of a run is not measured as a scalar, but as an attainment surface in m_f-dimensional space (where m_f is the number of objectives) [112]. A summary attainment surface (s.a.s) is defined as the union of all tightest goals that have been attained (independently) in precisely s of the runs of a sample of n runs, for any $s \in 1, \cdots, n$. This expresses the performance in terms of the quality attained in a certain fraction of sample runs. For example, the sample median quality is the best estimator of what one would expect to achieve in 50% of runs. Assume three independent runs are executed, and their corresponding attainment surfaces are plotted in Fig. 3.15 (a). Then, the best, median and worst s.a.s over the three independent runs can be illustrated in Fig. 3.15. For convenience, the corner where the line changes from vertical to horizontal is named as convex corner. The vertex of the convex corner, named convex corner point, is corresponding to the non-dominated solution found by the algorithm. The corner where the line transfers from horizontal to vertical is called concave corner. The concave corner shows the gap where the algorithm can not find any solution.

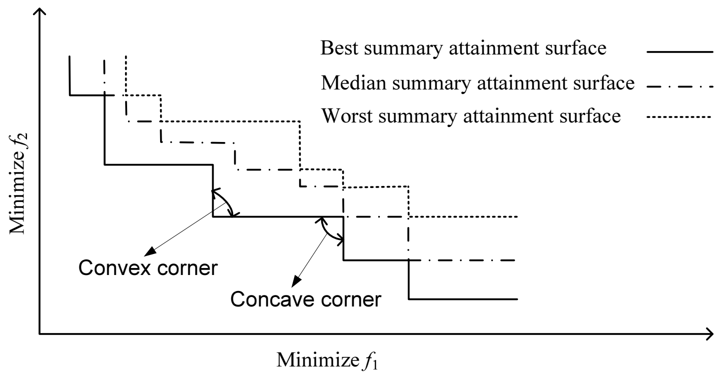
Comparing to the way of simply plotting solution points, using s.a.s can display the outcome of multiple runs of one optimiser. With the plot of the attainment surfaces, it is much easier to identify 'gaps' ($i.e.$ concave corners) in the distribution of solutions, hence, it is easier to interpret results correctly. It may be noticed that the s.a.s emphasizes the distribution of solutions, and also indicates the quality of the individual solutions.

3.4.2 Simulation results

In this section, FOLA is fully compared with two Pareto front-based algorithms, a promising multi-objective evolutionary algorithm based on decomposition (MOEA/D) [103], which was ranked first in the unconstrained MOEA competition [113], and non-dominated sorting genetic algorithm II (NSGA-II) [29]. The comparison is based on three groups of multi-objective benchmark functions, which represent a wide range of challenging multi-objective optimisation problems. The parameter settings of MOEA/D and NSGA-II follow the suggestions in [103] and [29] respectively. The parameters of MOLA are the same as those employed in FOLA except for $D = 20$, $M = 50$ and N_{femax}.



(a)


Best summary attainment surface
Median summary attainment surface
Worst summary attainment surface
Convex corner
Concave corner
Minimize f_2
Minimize f_1

(b)

Figure 3.15: The illustration of s.a.s. (a) Attainment surfaces of three independent runs; (b) s.a.s. obtained from the three runs

Multi-objective benchmark functions: Group 1

The first group of multi-objective benchmark functions includes two benchmark functions and their transformed formats, as given in Appendix A.5. The performance of MOLA in solving these functions may reflect its applicability for different cases encountered in practise. The first set of functions, denoted as Fun1, tests the ability of the algorithms in solving different scales of non-convex problems. Fun2 is to investigate the affect of problem characteristics (with respect to the symmetry of the two objective functions) on the performance of the algorithms.

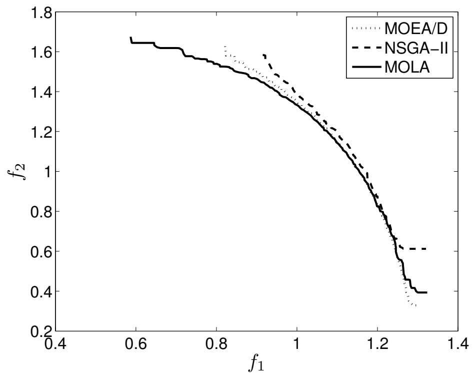
Function 1

Fun1 [27] has three different dimensional scales: 30-dimensional Fun1-1, 50-dimensional Fun1-2 and 70-dimensional Fun1-3, as given in Appendix A.5. For MOEA/D and MOLA, $N_{\text{femax}} = 15,000$ is employed to solve Fun1-1, while $N_{\text{femax}} = 25,000$ is set for Fun1-2, and $N_{\text{femax}} = 30,000$ is set for Fun1-3. As for NSGA-II, $N_{\text{femax}} = 150,000$, $N_{\text{femax}} = 250,000$ and $N_{\text{femax}} = 300,000$ are set for Fun1-1, Fun1-2 and Fun1-3 respectively.

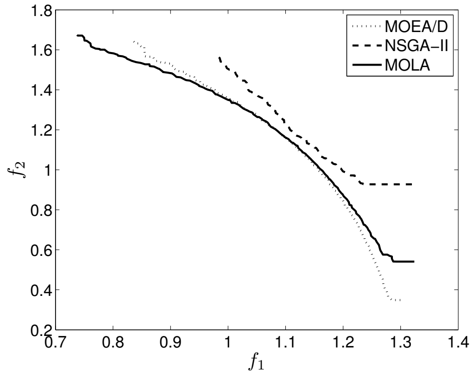
The extreme solutions are estimated by adopting the boundary solutions of the estimated Pareto front which is composed of all the non-dominated solutions found by the three algorithms. The reference point can be found by constructing a vector of the worst objective fitness values found by the three algorithms. Table 3.3 lists the setting of the reference solution and extreme solutions adopted, together with the results (including the mean and standard deviation of the obtained HV and Δ) obtained by the three algorithms in solving Fun1. Distance metric is not employed in this case, as the true Pareto front is unknown. It can be seen that MOLA can constantly find larger mean HV than MOEA/D and NSGA-II in solving the three problems; additionally, MOLA can find much smaller mean Δ than the other two algorithms. In some sense, this fact suggests that MOLA can find more accurate non-dominated solutions than MOEA/D and NSGA-II, and furthermore, the solutions found by MOLA is widely and uniformly spread. To visually observe both the strengths and weaknesses of the three algorithms, the best, median and worst s.a.s. over 30 independent runs are given in Figs. 3.16, 3.17 and 3.18, which are for Fun1-1, Fun1-2 and Fun1-3 respectively. In Fig. 3.16, MOLA finds better Pareto fronts than MOEA/D and NSGA-II in terms of the best s.a.s. With respect to the median s.a.s., MOLA finds better non-dominated solutions than MOEA/D when $f_1 < 1.1$. As for the worst s.a.s., MOEA/D obtains better Pareto front than MOLA. As the dimensionality increases, the superiority of MOLA over MOEA/D and NSGA-II is obvious, especially when $N = 70$, as given in Fig. 3.18, where MOLA outperforms MOEA/D and NSGA-II greatly with respect to the best, median and worst s.a.s.

Table 3.3: The setting of reference solution and extreme solutions; HV and Δ obtained by MOEA/D, NSGA-II and MOLA on Fun1 (including mean and standard deviation)

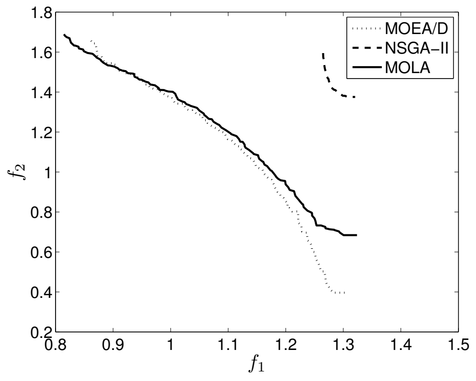
Fun1-1	Setting	Reference solution	Extreme solution 1		Extreme solution 2	
		[1.31 1.69]	[0.5868 1.6696]		[1.285 0.3291]	
	Results		HV		Δ	
			Mean	Std	Mean	Std
		MOEA/D	0.2671	0.0035	0.0119	0.0022
		NSGA-II	0.1564	0.0822	0.0146	0.0027
		MOLA	0.2702	0.0262	0.0061	0.0019
Fun1-2	Setting	Reference solution	Extreme solution 1		Extreme solution 2	
		[1.38 1.9]	[0.5843 1.87]		[1.3721 0.3627]	
	Results		HV		Δ	
			Mean	Std	Mean	Std
		MOEA/D	0.2741	0.0146	0.0156	0.0012
		NSGA-II	0.1452	0.0776	0.0178	0.0025
		MOLA	0.3021	0.0250	0.0078	0.0008
Fun1-3	Setting	Reference solution	Extreme solution 1		Extreme solution 2	
		[1.63 2.64]	[0.6564 2.0305]		[1.432 0.4322]	
	Results		HV		Δ	
			Mean	Std	Mean	Std
		MOEA/D	0.9007	0.0328	0.0164	0.0021
		NSGA-II	0.2574	0.2931	0.0252	0.0067
		MOLA	1.1837	0.0490	0.0043	0.0012



(a) Best s.a.s.

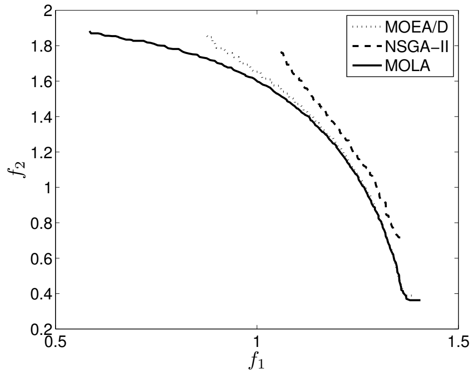


(b) Median s.a.s.

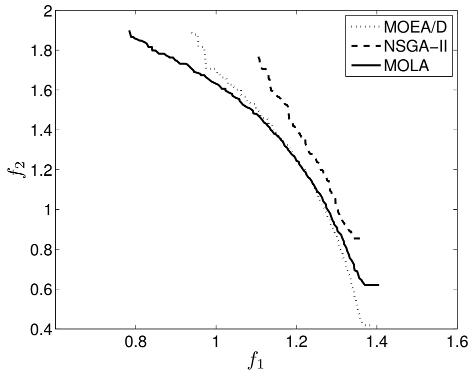


(c) Worst s.a.s.

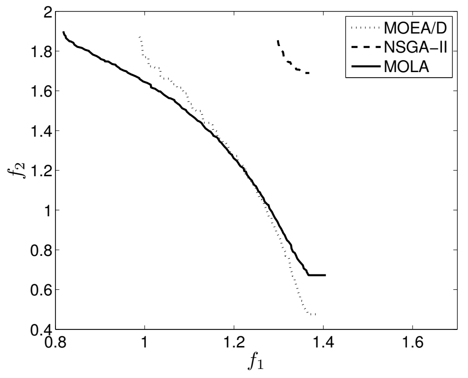
Figure 3.16: s.a.s. obtained by MOEA/D, NSGA-II and MOLA on Fun1-1



(a) Best s.a.s.

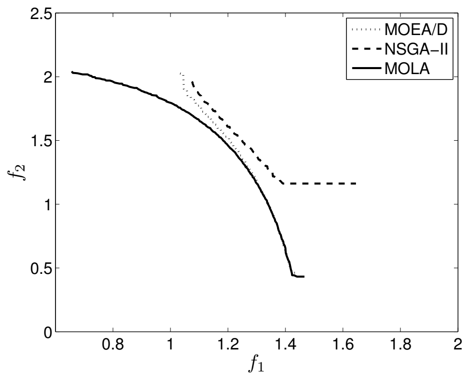


(b) Median s.a.s.

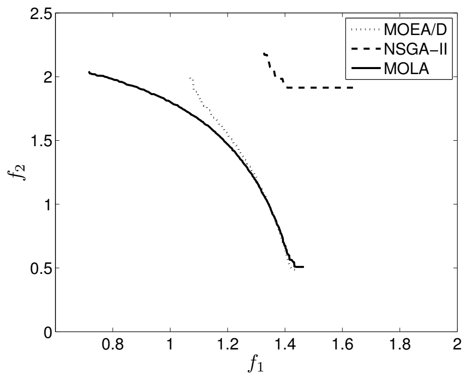


(c) Worst s.a.s.

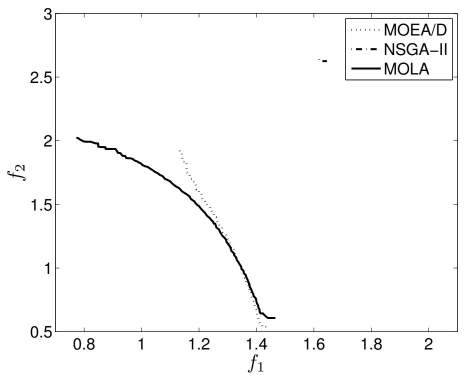
Figure 3.17: s.a.s. obtained by MOEA/D, NSGA-II and MOLA on Fun1-2



(a)Best s.a.s.



(b) Median s.a.s.



(c) Worst s.a.s.

Figure 3.18: s.a.s. obtained by MOEA/D, NSGA-II and MOLA on Fun1-3

Function 2

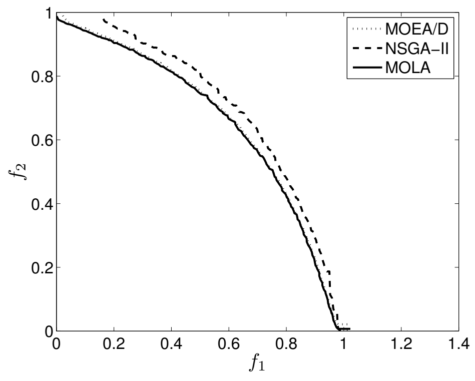
The original form of the set of problems is Fun2-1, which was proposed by Fonseca *et al.* [108]. The main feature of Fun2-2 is that its two objective functions have different ranges of fitness values. For Fun2-3, the ranges of the two objective functions are the same, however, the two objective functions are unsymmetric, as one of the objective function has the power of 2, and the other has the power of 6. To solve this set of functions, $N_{\text{femax}} = 15,000$ is employed by MOEA/D and MOLA, and $N_{\text{femax}} = 150,000$ is set for NSGA-II.

Table 3.4 lists the reference solution and extreme solutions that are employed in calculating HV and Δ respectively. The mean and standard deviation of HV and Δ obtained over 30 independent runs in solving Fun2 are also listed in Table 3.4. MOLA can find larger mean HV than both MOEA/D and NSGA-II in solving the three cases. Besides that, MOLA can find smaller mean Δ than MOEA/D, with the exception of Fun2-1, in which MOLA and MOEA/D find the equivalent mean and standard deviation of Δ.

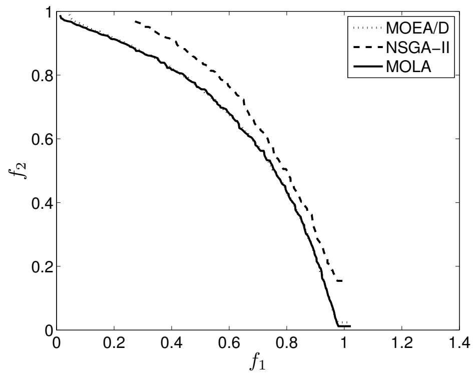
The s.a.s. obtained by the three algorithms on Fun2-1, Fun2-2 and Fun2-3 are given in Figs. 3.19, 3.20 and 3.21 respectively. For Fun2-1, MOLA and MOEA/D find similar s.a.s., while the surfaces found by NSGA-II are not accurate comparing to those obtained by MOEA/D and MOLA. As for Fun2-2, MOLA and MOEA/D also find similar solutions when $f_1 > 0.6$, however, the solutions obtained by MOLA are slightly better than those found by MOEA/D when $f_1 < 0.6$. In this case, NSGA-II can also find acceptable solutions, but these solutions are worse than those found by MOEA/D and MOLA. MOLA outperforms MOEA/D and NSGA-II obviously in solving Fun2-3. It can be seen from Fig. 3.21 (a) and (b) that the performance of MOEA/D degrades when $f_1 > 0.2$, in terms of the accuracy and smoothness of the obtained surfaces. MOLA can still perform well on this function, except for the worst s.a.s., on which the non-dominated solutions found by MOLA are slightly worse than those of MOEA/D when $f_1 < 0.2$. For this set of functions, the best and median s.a.s. found by NSGA-II are acceptable, however, its worst s.a.s. deviates widely from the best s.a.s. It suggests that the results obtained by NSGA-II vary widely over different independent runs.

Table 3.4: The setting of reference solution and extreme solutions; HV and Δ obtained by MOEA/D, NSGA-II and MOLA on Fun2 (including mean and standard deviation)

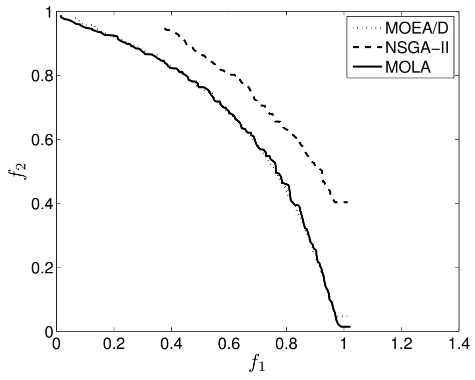
Fun2-1	Setting	Reference solution	Extreme solution 1	Extreme solution 2
		[0.985 0.984]	[7.504×10^{-4} 0.9821]	[0.9809 0.0022]
	Results		HV Mean Std	Δ Mean Std
		MOEA/D	0.2948 0.0019	0.0049 0.0011
		NSGA-II	0.2156 0.0340	0.0100 0.0010
		MOLA	0.2972 0.0064	0.0049 0.0011
Fun2-2	Setting	Reference solution	Extreme solution 1	Extreme solution 2
		[0.99 2.47]	[4.2805×10^{-4} 2.4586]	[0.9819 0.2025]
	Results		HV Mean Std	Δ Mean Std
		MOEA/D	0.6822 0.0094	0.0124 0.0004
		NSGA-II	0.5146 0.1477	0.0199 0.0027
		MOLA	0.7061 0.0130	0.0086 0.0021
Fun2-3	Setting	Reference solution	Extreme solution 1	Extreme solution 2
		[1 0.08]	[2.3826×10^{-4} 0.069]	[0.9778 1.778×10^{-11}]
	Results		HV Mean Std	Δ Mean Std
		MOEA/D	0.0666 0.0008	0.0106 0.0003
		NSGA-II	0.0632 0.0090	0.0057 0.0018
		MOLA	0.0707 0.0008	0.0072 0.0022



(a)Best s.a.s.

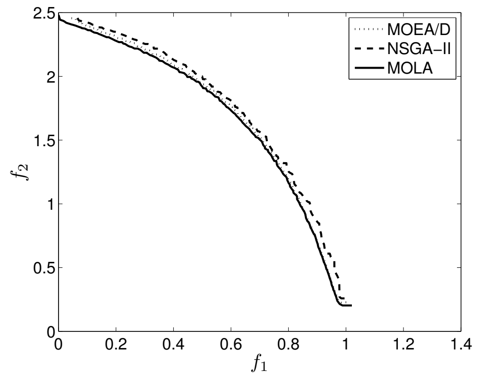


(b) Median s.a.s.

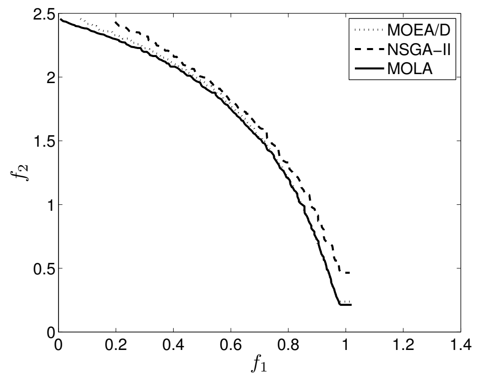


(c) Worst s.a.s.

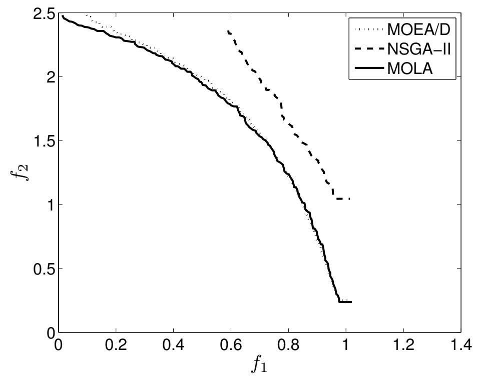
Figure 3.19: s.a.s. obtained by MOEA/D, NSGA-II and MOLA on Fun2-1



(a) Best s.a.s.

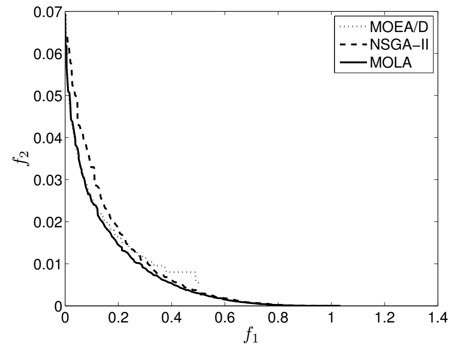


(b) Median s.a.s.

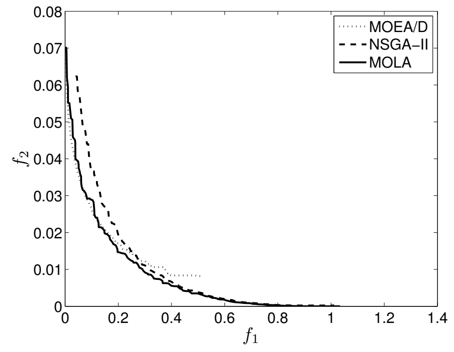


(c) Worst s.a.s.

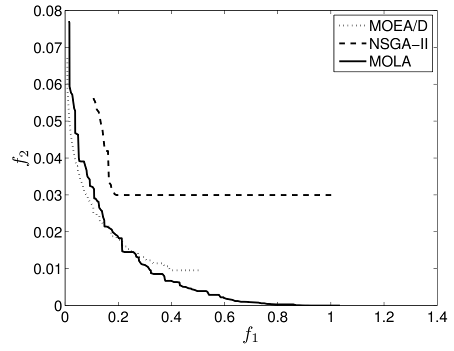
Figure 3.20: s.a.s. obtained by MOEA/D, NSGA-II and MOLA on Fun2-2



(a) Best s.a.s.



(b) Median s.a.s.



(c) Worst s.a.s.

Figure 3.21: s.a.s. obtained by MOEA/D, NSGA-II and MOLA on Fun2-3

Multi-objective benchmark functions: Group 2

This set of benchmark functions, as given in Appendix A.5, was used as multi-objective optimisation benchmark functions for CEC special session and competition [3] [114]. These functions, having complicated Pareto sets, are applied to investigate the impact of the Pareto set shapes on the performance of these algorithms, and evaluate the abilities of the algorithms to deal with complicated Pareto set shapes. Apart from HV and Δ, $\tilde{D}$ is also used to measure the performance of MOEA/D, NSGA-II and MOLA, as the true Pareto fronts of the problems are know. Since the ranges of the objective functions are the same for all the problems, $i.e.$ $[0, 1]$, reference solution is set to $[1, 1]$ for all the problems; and the two extreme solutions are set to $[0, 1]$ and $[1, 0]$. The same $N_{\text{femax}} = 300,000$ is employed by the three algorithms.

Table 3.5 lists the mean and standard deviation of HV, Δ and $\tilde{D}$ obtained by MOEA/D, NSGA-II and MOLA in solving the benchmark functions of group 2. Figs. 3.22-3.28 show the best, median and worst s.a.s. obtained by the three algorithms in solving the functions of group 2. In these figures, the true Pareto fronts are also provided, in order to make the comparison convenient.

In terms of the mean of HV, MOLA performs better than MOEA/D and NSGA-II on Fun4-Fun9, and has equal performance as MOEA/D on Fun3. In addition, the standard deviation of HV obtained by MOLA is much smaller than that obtained by MOEA/D and NSGA-II.

The distance metric obtained by MOLA is smaller than MOEA/D and NSGA-II on these seven functions with the exception of Fun8, on which MOEA/D has a smaller value. The reason that the distance metric obtained by MOLA is larger than that of MOEA/D is because MOLA finds some non-dominated solutions above the disconnected area of the Pareto front, as given in Fig. 3.27. These non-dominated solutions make the distance metric obtained by MOLA larger than expected. It can be seen from the median s.a.s. (in Fig. 3.27 (a)) that the number of the non-dominated solutions found by MOEA/D is five, which can by counted by adding the number of the convex corner points and the two boundary solutions. Additionally, in Fig. 3.27, the three surfaces (including the best, median and worst s.a.s.) obtained by MOLA are quite similar, while those obtained by MOEA/D and NSGA-II vary widely. It means that the performance of MOLA is stable, in comparison with MOEA/D and NSGA-II.

As for Δ, MOLA finds smaller mean values than NSGA-II with the exception of Fun8. Compared with MOEA/D, MOLA performs better on these functions except for Fun6 and Fun8. In Fig. 3.25, the Pareto front found by MOLA is not smooth as that of MOEA/D, which explains the reason why MOLA has a larger mean value of Δ. However, it can be seen from Fig. 3.25 that the performance of MOLA is much better than MOEA/D in terms of the best, median and worst s.a.s., since the surfaces found by MOLA are much closer to the true Pareto front. This also explains why sometimes more than one performance measure is required for comprehensive comparison. As for Fun8, as analyzed above, the two disconnected gaps result in large values of d_i in (3.4.2), which makes Δ inappropriate as performance metric in this case.

To visually compare the performance of the three algorithms over a number of independent runs, their s.a.s. in solving these functions (with the exception of Fun6 and Fun8 that have been discussed above) are discussed in detail as follows:

- For Fun3, it can be seen from Fig. 3.22 that MOLA and MOEA/D can find the best and median s.a.s. which have similar shapes and location as the true Pareto front. The worst s.a.s. found by MOLA is also very accurate compared with the true Pareto front. As for MOEA/D, its worst s.a.s. is also very close to the true Pareto front except for the small area where $f_2 > 0.8$. The s.a.s. (including the best, median, and worst ones) found by NSGA-II are not so accurate compared with those found by MOEA/D and MOLA.

- For Fun4, as given in Fig. 3.23, the best and median s.a.s. found by MOLA are almost the same as the true Pareto front, while the worst s.a.s. is inaccurate when $f_1 > 0.9$. As for MOEA/D, its best s.a.s. is close to the true Pareto front. However, there are gaps ($i.e.$ concave corners) existing on the median and worst s.a.s. when $f_1 > 0.8$. It suggests that MOEA/D cannot find any non-dominated solution locating in this area. As for NSGA-II, its median s.a.s. is inaccurate when $f_1 > 0.7$, and its worst s.a.s. is inaccurate when $f_1 > 0.6$.

- The s.a.s. obtained by these algorithms on Fun5 are given in Fig. 3.24. The best and median s.a.s. found by MOEA/D and MOLA are quite accurate. For the worst s.a.s., MOLA performs better than MOEA/D. In this case, NSGA-II can only find a limited area of the true Pareto front.
- Fun7 has a discrete Pareto front, as shown in Fig. 3.25. It can be seen that MOLA can successfully solve this function, while MOEA/D and NSGA-II have poor performance.
- As for Fun9, MOLA and MOEA/D perform well in terms of the best, median and worst s.a.s., as shown in Fig. 3.28. NSGA-II can find approximately half of the Pareto front accurately, while its performance is poor when $f_1 > 0.5$, as given in the Fig. 3.28 (b).

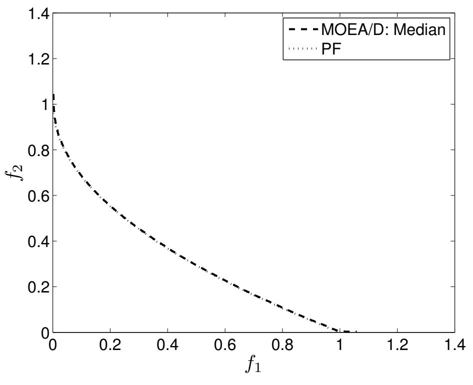
Multi-objective benchmark function: Group 3

Group 3 includes four benchmark functions, which have three objectives, as given in Appendix A.5. The first two functions have complicated Pareto set [114], and their dimensionality is set to 30. The latter two functions are known as functions DTLZ1 and DTLZ2 respectively [103], and they are 10-dimensional. The same $N_{\text{femax}} = 300,000$ is adopted by the three algorithms. The comparison of the three algorithms are carried out through the performance metric of $\tilde{D}$. Table 3.6 lists the mean and standard deviation of $\tilde{D}$. It can be seen that MOLA can constantly find smaller mean $\tilde{D}$ than MOEA/D and NSGA-II.

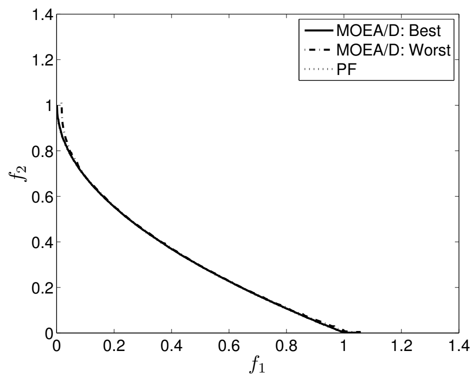
Since $\tilde{D}$ cannot reveal the spreadout of the Pareto solutions, s.a.s. is also employed to observe the smoothness and spreadout of the Pareto fronts found by the three algorithms. In order to observe the s.a.s. clearly, the coordination of f_3 is reversed, as given in Figs. 3.29-3.32, which show the s.a.s obtained by the algorithms when solving the benchmark functions of group 3. The s.a.s. found by NSGA-II on Fun10-Fun12 are inacceptable. NSGA-II has better performance in solving Fun13,

Table 3.5: $\tilde{D}$, HV and Δ obtained by MOEA/D, NSGA-II and MOLA on Fun3-Fun9 (including mean and standard deviation)

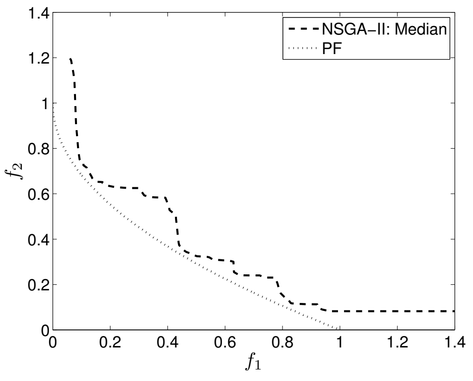
		$\tilde{D}$		HV		Δ	
		Mean	Std	Mean	Std	Mean	Std
Fun3	MOEA/D	5.8193×10^{-6}	3.0964×10^{-6}	0.6628	0.0011	0.0013	8.1690×10^{-5}
	NSGA-II	0.0220	0.0286	0.5516	0.0449	0.0193	0.0058
	MOLA	5.1015×10^{-6}	1.5555×10^{-6}	0.6628	8.1822×10^{-4}	0.0012	1.6661×10^{-4}
Fun4	MOEA/D	1.9535×10^{-4}	1.7043×10^{-4}	0.6558	0.0051	0.0019	1.0105×10^{-4}
	NSGA-II	0.0011	8.4128×10^{-4}	0.6280	0.0045	0.0113	0.0021
	MOLA	7.6169×10^{-6}	3.3100×10^{-6}	0.6608	8.5643×10^{-4}	0.0010	1.0690×10^{-4}
Fun5	MOEA/D	8.4750×10^{-5}	1.1381×10^{-4}	0.6555	0.0102	0.0011	2.2503×10^{-4}
	NSGA-II	0.0379	0.0156	0.3868	0.0415	0.0204	0.0024
	MOLA	4.0240×10^{-6}	5.8248×10^{-7}	0.6611	0.0014	4.6123×10^{-4}	7.2860×10^{-5}
Fun6	MOEA/D	0.0046	6.4575×10^{-4}	0.2442	0.0043	0.0024	1.8647×10^{-4}
	NSGA-II	0.0069	3.4784×10^{-4}	0.2267	0.0038	0.0143	6.7051×10^{-4}
	MOLA	4.9992×10^{-4}	4.1571×10^{-5}	0.2976	0.0011	0.0053	5.5715×10^{-4}
Fun7	MOEA/D	0.0897	0.0239	0.0312	0.0237	0.0149	0.0074
	NSGA-II	7.7018	4.0972	0	0	0.1047	0.0415
	MOLA	8.9431×10^{-4}	4.9639×10^{-4}	0.4585	5.5918×10^{-4}	0.0081	0.0013
Fun8	MOEA/D	0.0180	0.0120	0.2404	0.0630	0.0145	0.0132
	NSGA-II	0.0191	0.0124	0.1519	0.0859	0.0229	0.0088
	MOLA	0.0330	0.0069	0.3013	0.0132	0.0235	0.0023
Fun9	MOEA/D	1.5085×10^{-5}	1.6198×10^{-5}	0.4962	8.3737×10^{-4}	0.0013	3.2527×10^{-4}
	NSGA-II	0.0214	0.0207	0.3200	0.1822	0.0185	0.0072
	MOLA	3.7590×10^{-6}	1.3175×10^{-6}	0.4966	4.9602×10^{-4}	8.3782×10^{-4}	1.4161×10^{-4}



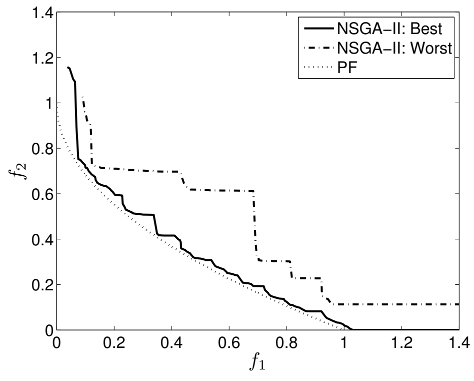
(a) Median s.a.s of MOEA/D



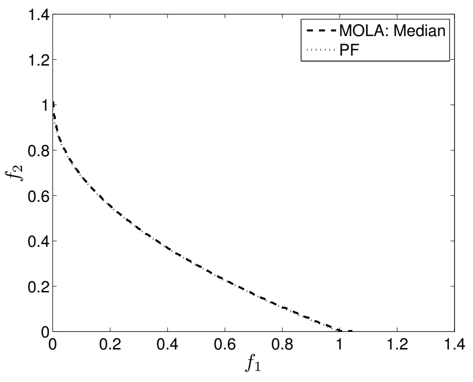
(b) Best and worst s.a.s of MOEA/D



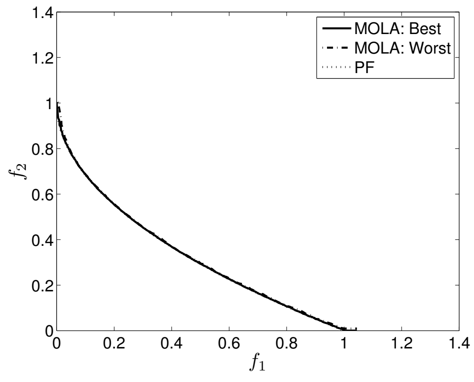
(c) Median s.a.s of NSGA-II



(d) Best and worst s.a.s of NSGA-II

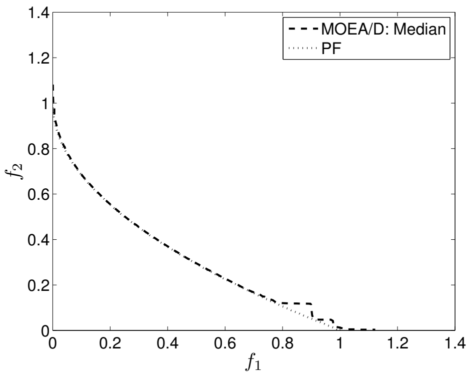


(e) Median s.a.s of MOLA

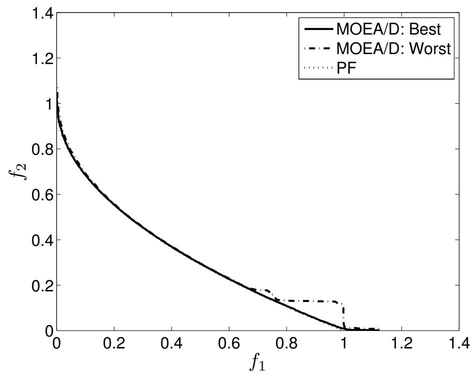


(f) Best and worst s.a.s of MOLA

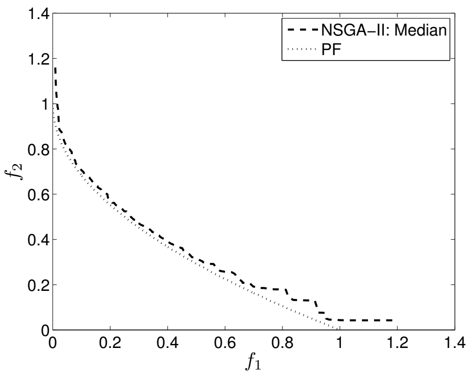
Figure 3.22: s.a.s obtained by MOEA/D, NSGA-II and MOLA on Fun3



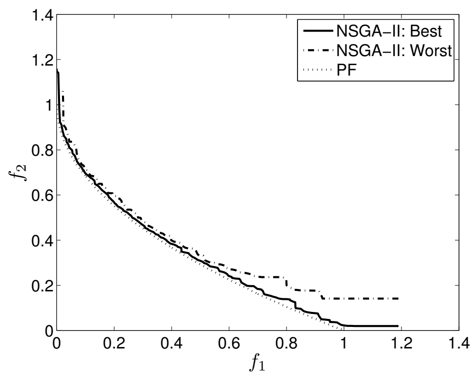
(a) Median s.a.s of MOEA/D



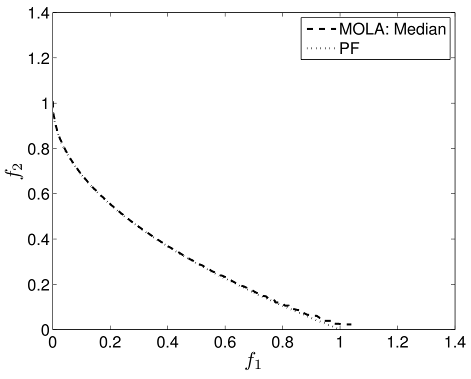
(b) Best and worst s.a.s of MOEA/D



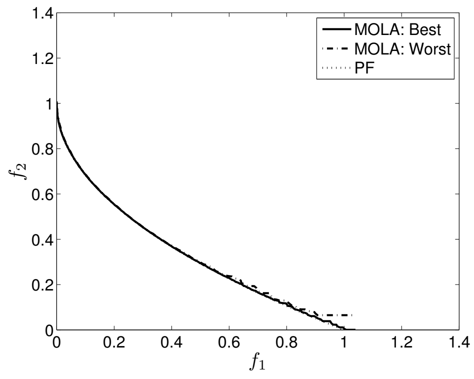
(c) Median s.a.s of NSGA-II



(d) Best and worst s.a.s of NSGA-II

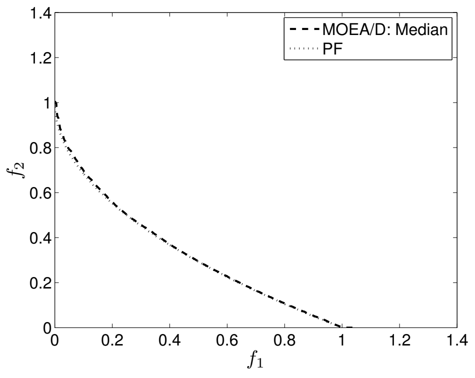


(e) Median s.a.s of MOLA

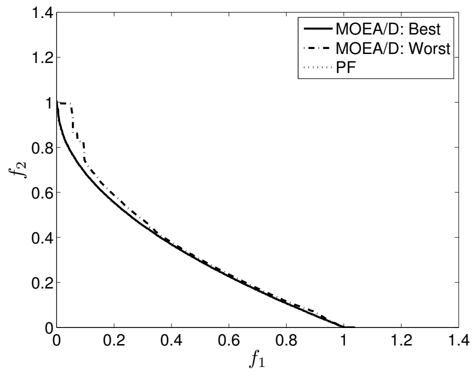


(f) Best and worst s.a.s of MOLA

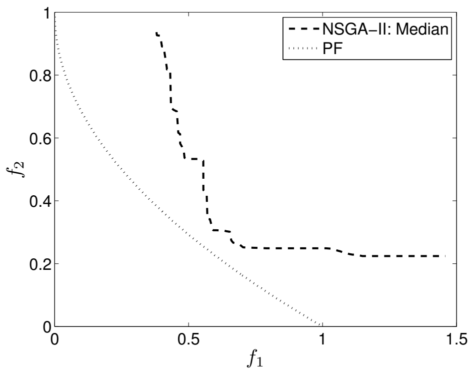
Figure 3.23: s.a.s obtained by MOEA/D, NSGA-II and MOLA on Fun4



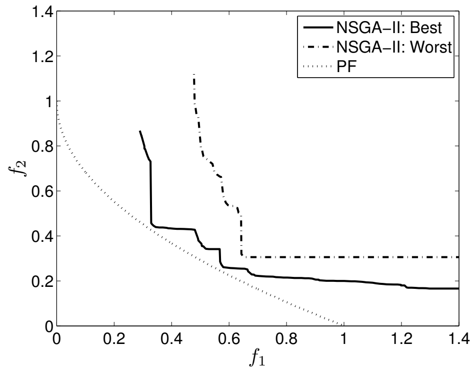
(a) Median s.a.s of MOEA/D



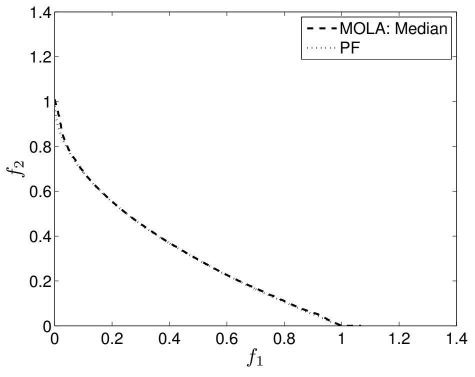
(b) Best and worst s.a.s of MOEA/D



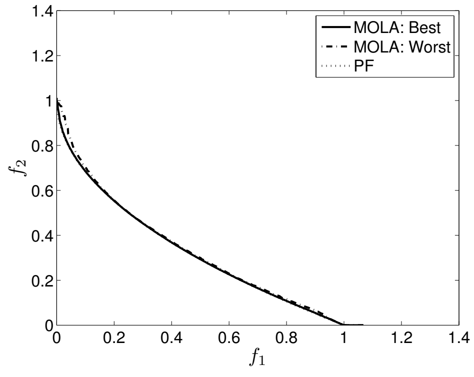
(c) Median s.a.s of NSGA-II



(d) Best and worst s.a.s of NSGA-II

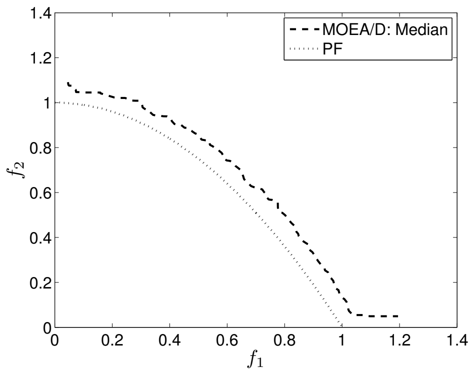


(e) Median s.a.s of MOLA

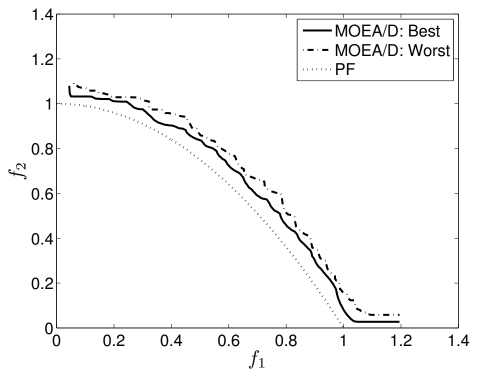


(f) Best and worst s.a.s of MOLA

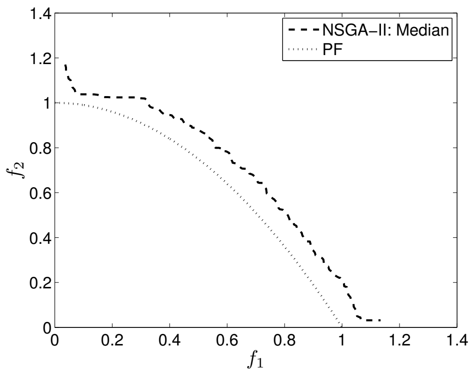
Figure 3.24: s.a.s obtained by MOEA/D, NSGA-II and MOLA on Fun5



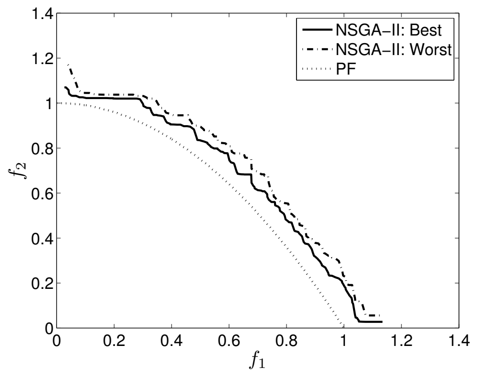
(a) Median s.a.s of MOEA/D



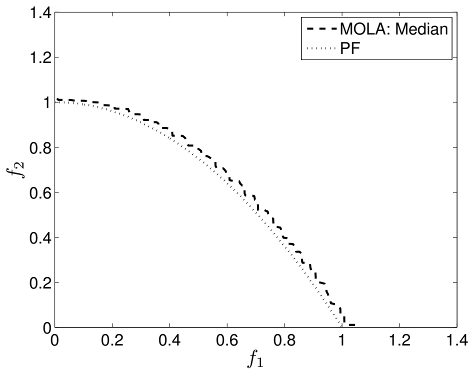
(b) Best and worst s.a.s of MOEA/D



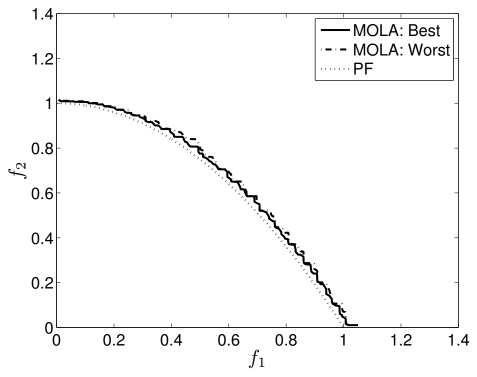
(c) Median s.a.s of NSGA-II



(d) Best and worst s.a.s of NSGA-II

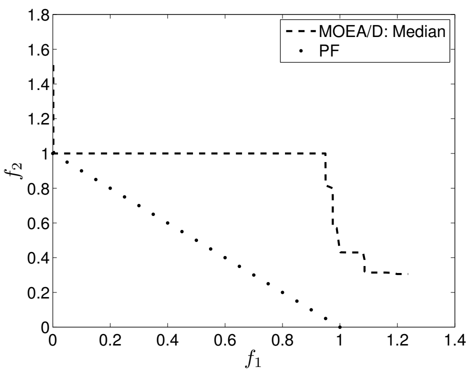


(e) Median s.a.s of MOLA

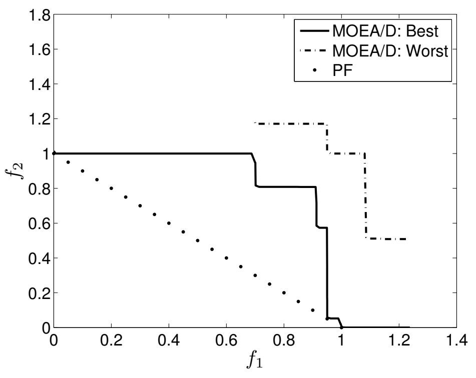


(f) Best and worst s.a.s of MOLA

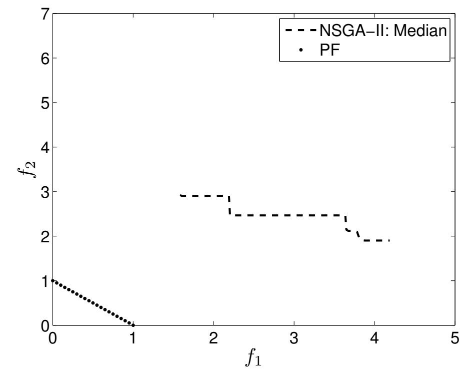
Figure 3.25: s.a.s obtained by MOEA/D, NSGA-II and MOLA on Fun6



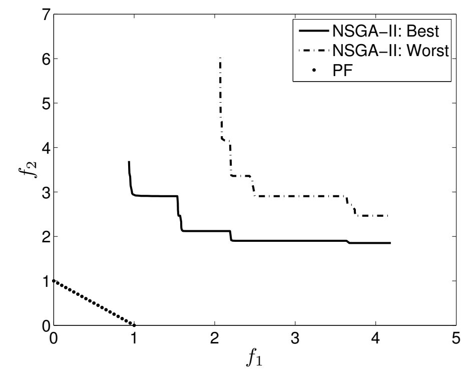
(a) Median s.a.s of MOEA/D



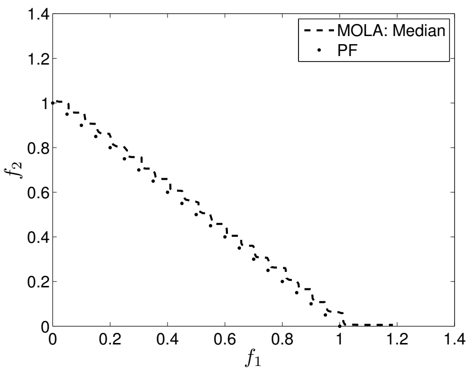
(b) Best and worst s.a.s of MOEA/D



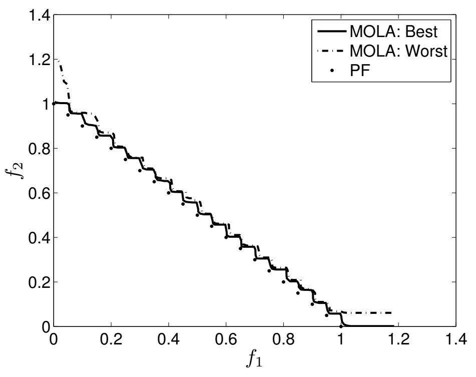
(c) Median s.a.s of NSGA-II



(d) Best and worst s.a.s of NSGA-II

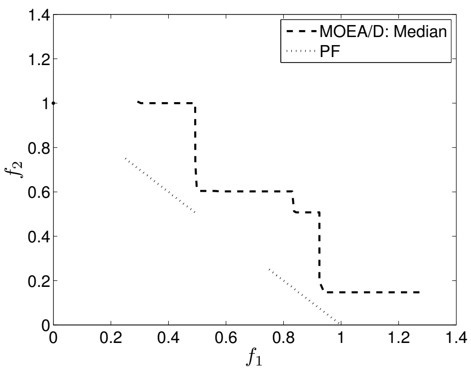


(e) Median s.a.s of MOLA

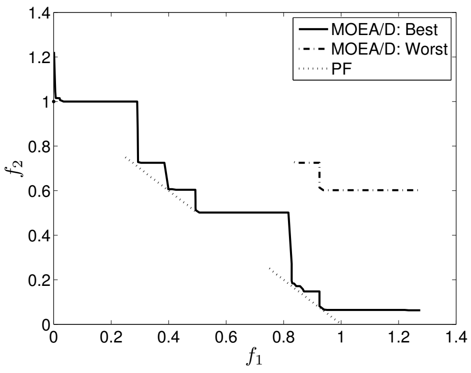


(f) Best and worst s.a.s of MOLA

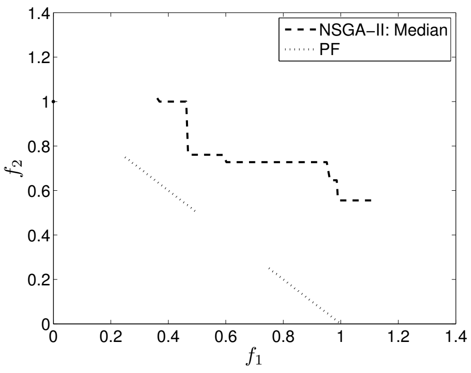
Figure 3.26: s.a.s obtained by MOEA/D, NSGA-II and MOLA on Fun7



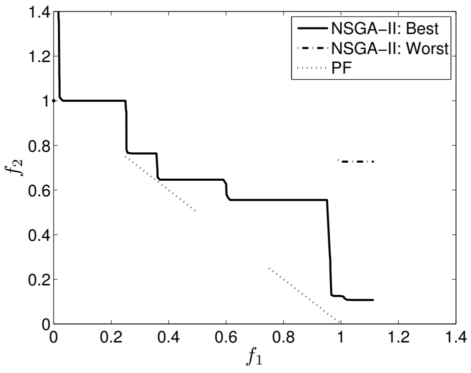
(a) Median s.a.s of MOEA/D



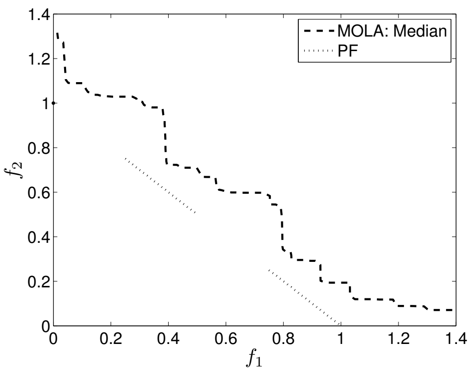
(b) Best and worst s.a.s of MOEA/D



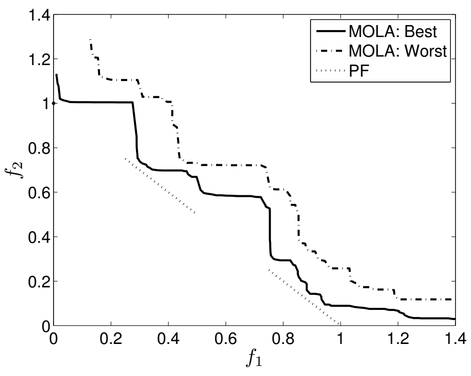
(c) Median s.a.s of NSGA-II



(d) Best and worst s.a.s of NSGA-II

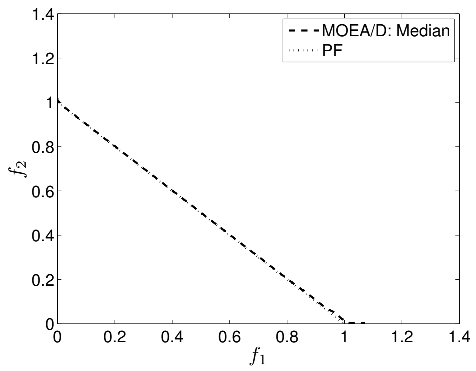


(e) Median s.a.s of MOLA

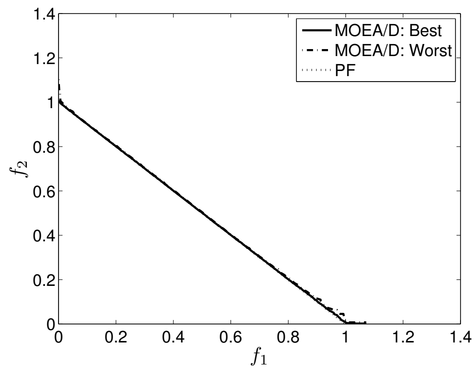


(f) Best and worst s.a.s of MOLA

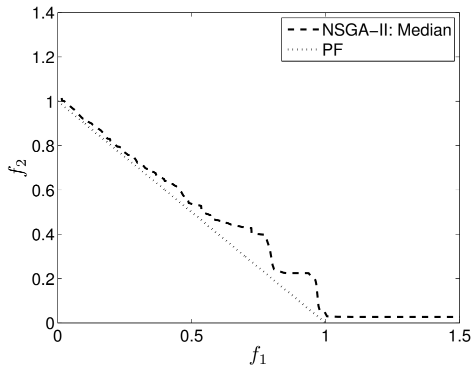
Figure 3.27: s.a.s obtained by MOEA/D, NSGA-II and MOLA on Fun8



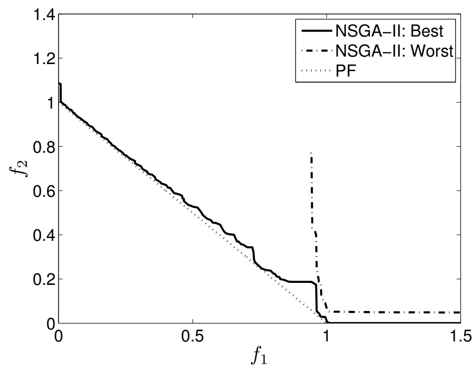
(a) Median s.a.s of MOEA/D



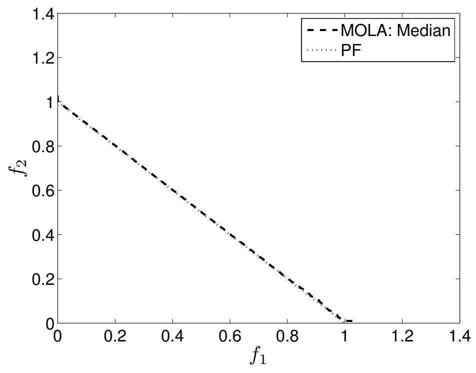
(b) Best and worst s.a.s of MOEA/D



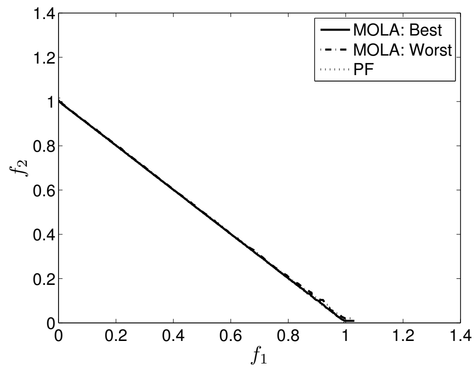
(c) Median s.a.s of NSGA-II



(d) Best and worst s.a.s of NSGA-II



(e) Median s.a.s of MOLA



(f) Best and worst s.a.s of MOLA

Figure 3.28: s.a.s obtained by MOEA/D, NSGA-II and MOLA on Fun9

however, the obtained surfaces are not smooth. Since NSGA-II has poor performance in solving this set of functions, the comparison is mainly carried out between MOLA and MOEA/D, as discussed in the following:

- For Fun10, the true Pareto front is $f_1^2+f_2^2+f_3^3=1$, where $0 \leq f_1, f_2, f_3 \leq 1$. Fig. 3.29 provides the best, median and worst s.a.s. found by MOEA/D, NSGA-II and MOLA. In terms of the best and median s.a.s., MOEA/D cannot find the solutions that locate at $f_1 < 0.4$ and $f_2 < 0.6$, which can be revealed by the gaps in the figure; MOLA can find the solutions which cover the whole true Pareto front. As for the worst s.a.s., most of the area of the true Pareto front cannot be found by MOEA/D, while MOLA can find around half of them.

- The true Pareto front of Fun11 includes two separate triangle surfaces: the first triangle surface can be formulated as $0 \leq f_3 \leq 1$, $0 \leq f_1 \leq \frac{1}{4}(1-f_3)$, $f_2 = 1-f_1-f_3$; while the second one is $0 \leq f_3 \leq 1$, $\frac{3}{4}(1-f_3) \leq f_1 \leq 1$, $f_2 = 1-f_1-f_3$. For this function, both MOEA/D and MOLA can find acceptable s.a.s., as given in Fig. 3.30. However, for the worst s.a.s., MOEA/D can only find one of the two triangle surfaces, while MOLA can still find both of the triangle surfaces.

- The true Pareto front of Fun12 is a big triangle surface $\sum_{i=1}^{3} f_i = 1$ with $f_i \geq 0$. In this case, MOLA can find smoother s.a.s.(including the best, median and worst ones) than MOEA/D, as shown in Fig. 3.31.

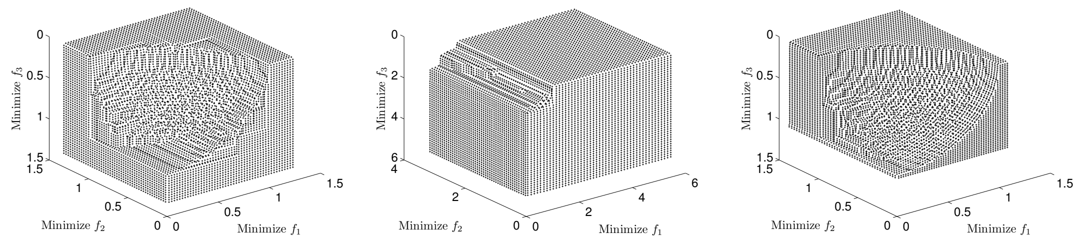
- As for Fun13, its true Pareto front is $\sum_{i=1}^{3} f_i^2 = 1$ with $f_i \geq 0$. It can be seen from Fig. 3.32 that the best, median and worst s.a.s. found by MOLA are smoother than those obtained by MOEA/D.

3.5 Conclusions

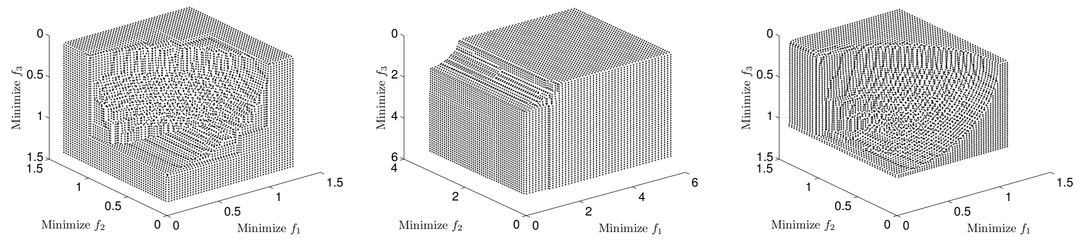
This chapter has presented a multi-objective optimisation by learning automata (MOLA), which capitalises on the merits of the structure of multiple automata, the

Table 3.6: $\tilde{D}$ obtained by NSGA-II, MOEA/D and MOLA on Fun10-Fun13 (including mean and standard deviation)

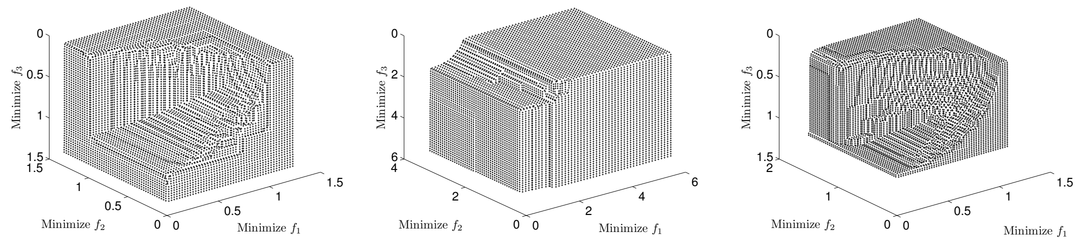
		$\tilde{D}$	
		Mean	Std
Fun10	MOEA/D	0.0016	6.4469×10^{-4}
	NSGA-II	3.8508	0.9978
	MOLA	0.0015	6.3562×10^{-4}
Fun11	MOEA/D	0.0167	0.0169
	NSGA-II	28.0860	13.0810
	MOLA	0.0035	0.0012
Fun12	MOEA/D	1.5771×10^{-4}	4.0956×10^{-5}
	NSGA-II	2.5456×10^{3}	1.0898×10^{3}
	MOLA	6.5282×10^{-5}	9.0417×10^{-5}
Fun13	MOEA/D	0.0076	4.8336×10^{-6}
	NSGA-II	0.0153	6.2305×10^{-4}
	MOLA	0.0048	0.0001



(a) Best s.a.s (MOEA/D); (b) Best s.a.s (NSGA-II); (c) Best s.a.s (MOLA)

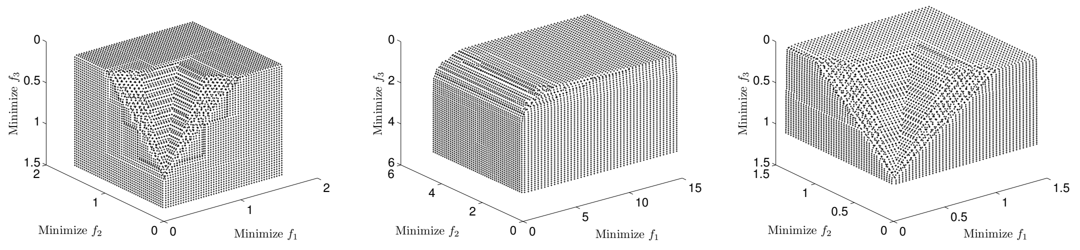


(d) Median s.a.s (MOEA/D); (e) Median s.a.s (NSGA-II); (f) Median s.a.s (MOLA)

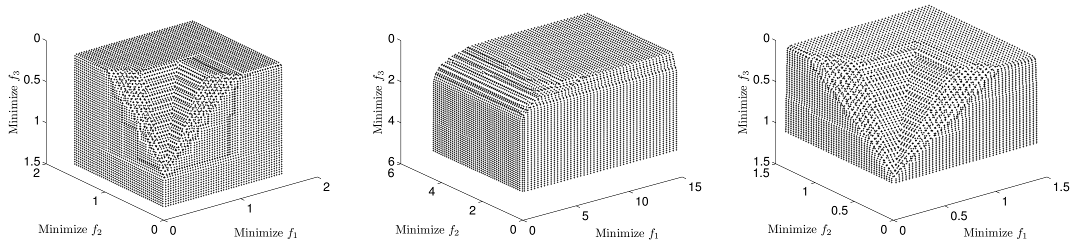


(g) Worst s.a.s (MOEA/D); (h) Worst s.a.s (NSGA-II); (i) Worst s.a.s (MOLA)

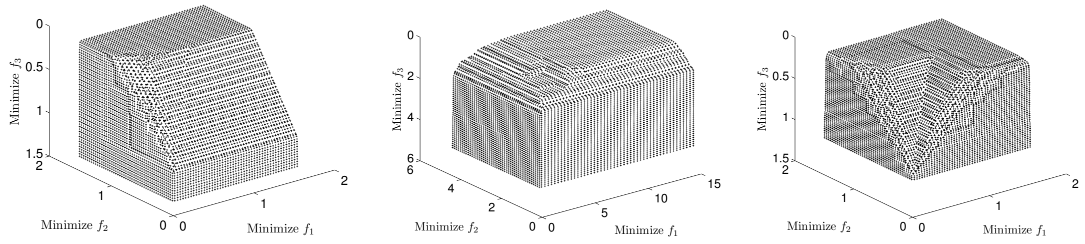
Figure 3.29: s.a.s obtained by MOEA/D, NSGA-II and MOLA on Fun10



(a) Best s.a.s (MOEA/D); (b) Best s.a.s (NSGA-II); (c) Best s.a.s (MOLA)

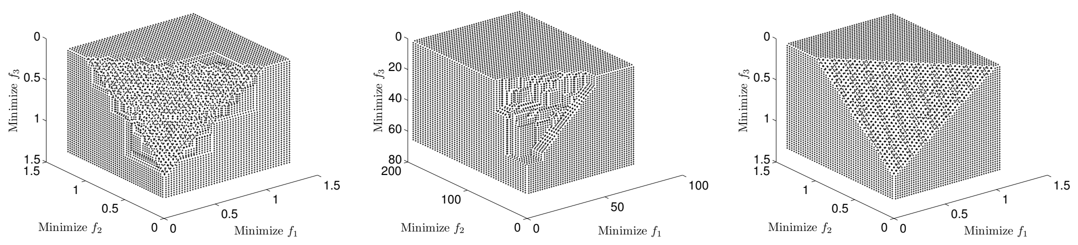


(d) Median s.a.s (MOEA/D); (e) Median s.a.s (NSGA-II); (f) Median s.a.s (MOLA)

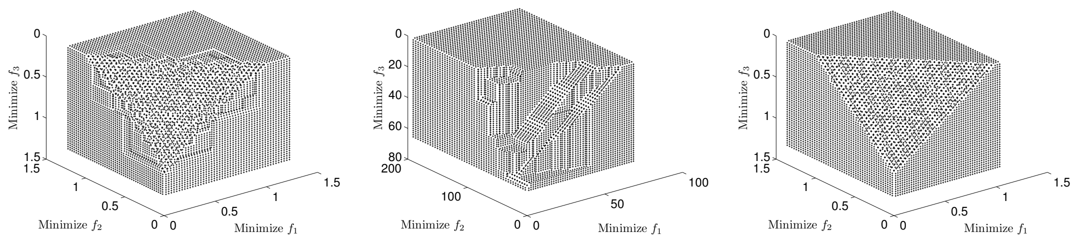


(g) Worst s.a.s (MOEA/D); (h) Worst s.a.s (NSGA-II); (i) Worst s.a.s (MOLA)

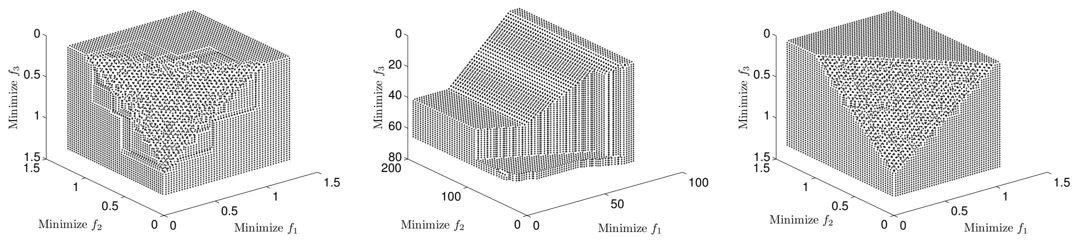
Figure 3.30: s.a.s obtained by MOEA/D, NSGA-II and MOLA on Fun11



(a) Best s.a.s (MOEA/D); (b) Best s.a.s (NSGA-II); (c) Best s.a.s (MOLA)

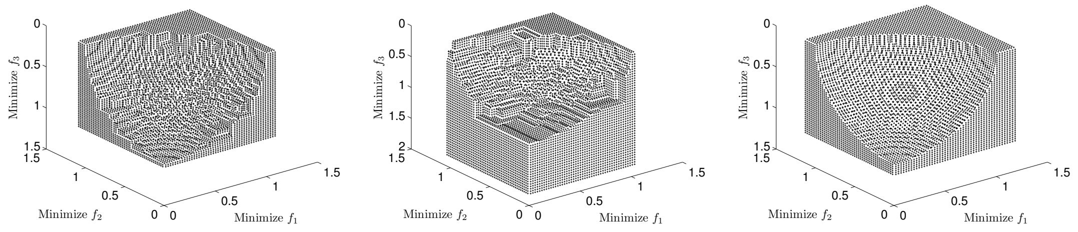


(d) Median s.a.s (MOEA/D); (e) Median s.a.s (NSGA-II); (f) Median s.a.s (MOLA)

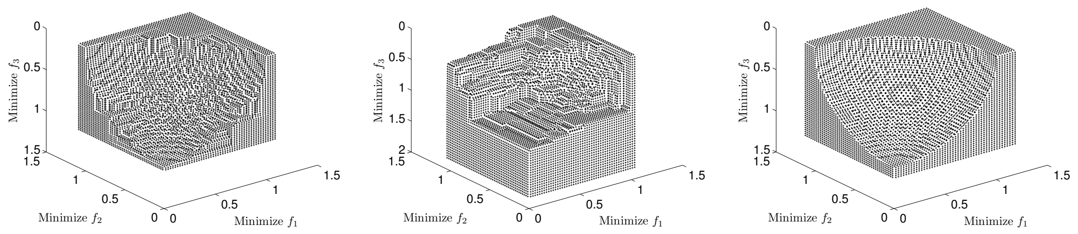


(g) Worst s.a.s (MOEA/D); (h) Worst s.a.s (NSGA-II); (i) Worst s.a.s (MOLA)

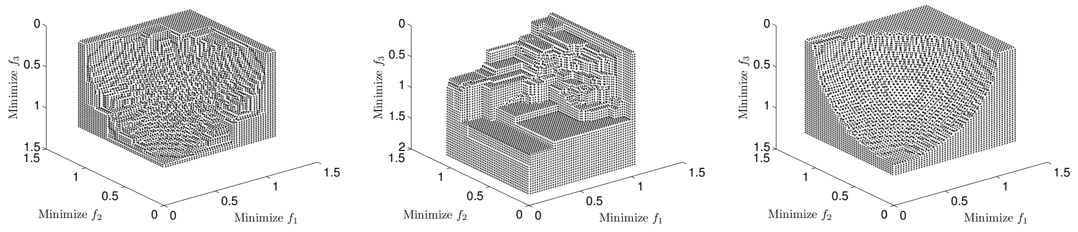
Figure 3.31: s.a.s obtained by MOEA/D, NSGA-II and MOLA on Fun12



(a) Best s.a.s (MOEA/D); (b) Best s.a.s (NSGA-II); (c) Best s.a.s (MOLA)



(d) Median s.a.s (MOEA/D); (e) Median s.a.s (NSGA-II); (f) Median s.a.s (MOLA)



(g) Worst s.a.s (MOEA/D); (h) Worst s.a.s (NSGA-II); (i) Worst s.a.s (MOLA)

Figure 3.32: s.a.s obtained by MOEA/D, NSGA-II and MOLA on Fun13

dimensional search, the dividing of the dimensional search domain into cells, the memories of the performance evaluation of the dimensional states in a form of cell values, and the learning and searching process. With these approaches, MOLA is able to undertaken search in continuous states, and achieve accurate Pareto sets and wide Pareto fronts efficiently.

MOLA has been fully compared with two popular weighted-sum based algorithms, MOGA and MOPSO, on four multi-objective benchmark functions that comprise low and high-dimensional models, convex and non-convex models, and continuous and discontinuous models respectively. MOLA finds more accurate non-dominated solutions than the other two algorithms. The solutions found by MOGA and MOPSO flock to the two ends of the Pareto fronts, when solving the problems which have non-convex Pareto front. As the dimensionality increases, MOLA exhibits great superiority over MOGA and MOPSO in terms of the quality of Pareto fronts and computation time. With less computation time, the solutions found by MOLA can dominate most of the solutions found by MOGA and MOPSO. MOLA has been also compared with two Pareto front-based multi-objective algorithms, MOEA/D and NSGA-II, on the basis of thirteen widely used multi-objective functions, which comprise complex Pareto set shapes. The simulation results have shown that MOLA greatly exhibits its superiority over MOEA/D and NSGA-II, as it can find more accurate and evenly distributed non-dominated solutions than MOEA/D and NSGA-II, and its Pareto fronts are wider than those obtained by MOEA/D and NSGA-II.

Part 2

Power System Applications Using Learning Automata-based Optimisation Algorithms

Chapter 4

The Application of FOLA on Optimal Power Flow Problems

4.1 Introduction

In power systems, various optimisation problems are getting more attention from researchers and engineers. For instance, power system dispatch problem has been intensively studied and widely used in power system operation and planning [115]. Voltage stability is another important factor in power systems, as voltage instability is a major power system weakness, which can result in severe detriments with economical, technical and social dimensions [116]. These concerns can be categorized as optimal power flow problems, which aim at achieving an optimal solution of a specific power system objective function, for instance, minimising the fuel cost, improving the voltage profile, and enhancing the voltage stability. The goal can be achieved by properly adjusting control variables, such as generator bus voltage magnitudes, generator real power outputs, transformer tap settings and reactive power of capacitor banks, and so on. The optimal power flow problem is a highly constrained complex optimisation problem, which has the nature of non-differential, non-linearity and non-convex. A variety of conventional methods have been applied to solve the problem, such as Nonlinear Programming [117] [118], Quadratic Programming [119], Linear Programming [120] and the Interior Point method [121]

[122], $etc.$ In most cases, the traditional methods are suitable for single-peak and linear objective functions. However, for solving the optimisaton problems with complex objective, these conventional methods suffer the drawback of easily trapping in local optima, thus cannot guarantee to find the global optimum solution. Various EAs introduced in Section 1.2.2 have attracted great attention and been applied to solve power system dispatch and voltage control problems, due to their ability to solve multi-modal optimisation problems [123] [124]. However EAs introduce redundant computation into the optimisation process and limit the computational capability, hence they are time-consuming when solving the high-dimensional optimisation problems, which are commonly encountered in power system applications. In this chapter, FOLA is applied to solve the problem of optimising power system dispatch and voltage stability.

With the increasing demand of power nowadays and the stress of the resources which can be used to generate power, renewable energy becomes more and more important in many countries, especially wind energy [125][126]. Wind power is widely used in Europe, Asia, and the United States, due to it is clean, reliable and quick to install. Although the dramatic improvements in the performance and affordability of wind turbines have paved the way for mass commercialization, wind power penetrated to power systems brings new challenges to power system economic operation. It is imperative to study how to efficiently solve optimal power flow problems which involve wind power, so as to achieve an optimal solution to the specific power system objective functions. The objective of the problem concerned in this chapter includes the minimisation of fuel cost and the enhancement of the voltage stability. The solution to the problem can be achieved by properly adjusting control variables such as generator real power outputs (including fuel generators and wind generators), generator bus voltage magnitudes, transformer tap settings and reactive power of capacitor banks. A variety of methods have been applied to solve optimal power flow problems in wind power penetrated systems. Q. Ai $et.\ al.$ [127] applied primal-dual interior point algorithm to reduce power loss in the transmission lines. However their method has only been investigated in a small-scale power system without comparison with the most popularly used EAs, such as GA, PSO, and Bacterial Foraging algorithm [128], which have been successfully applied to solve power system dispatch and voltage control problems [123] [124]. Besides, in [129], S. Brini $et.\ al.$ have used EAs to reduce the fuel cost and the emissions of the polluting gases. In this chapter, FOLA is presented to solve the optimisation problems concerned with economic power system dispatch and voltage stability enhancement, in wind power penetrated systems whose operation condition varies for a short period time. FOLA has been applied to solve the problems in the modified IEEE 30-bus, 57-bus and 118-bus power systems respectively, which are penetrated with dynamic wind power configuration which varies for a short period of time. FOLA is fully compared with popularly-used EAs. The simulation results have confirmed the effectiveness of FOLA in comparison with the popularly-used EAs, as FOLA can track the changing system configuration more rapidly and accurately.

4.2 Evaluation on Dispatch and Voltage Stability Enhancement Problems

4.2.1 Problem formulation

Optimal power flow problems (OPF)

The diagram of a typical electrical power system [130] is illustrated in Fig. 4.1. It can be seen that the typical power system consists of three parts: generating station generating electrical energy, transmission lines interconnecting the power system, and customers consuming the electrical energy.

The OPF problems, intensively studied as a network-constrained economic dispatch problem since its introduction by Carpentier [131] in 1962, can be formulated as a nonlinear constrained optimisation problem [115]. The problem is concerned with minimisation of a specific power system's objective function, by optimising control variables, within a set of operational and physical constraints. It can be

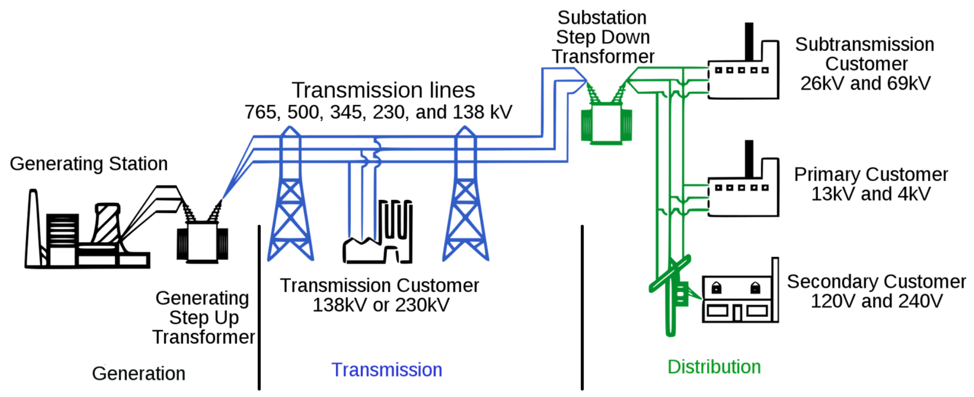


Figure 4.1: Diagram of an electrical system

formulated as follows:

$$\begin{aligned} &\min f(X,U) \\ &\texttt{s.t.}\ \ g(X,U)=0;\ h(X,U)\le 0. \end{aligned} \tag{4.2.1}$$

where g and h denote constraint functions [124][132]. X is a set of control variables to be optimised, including generator real power outputs P_{G} (with the exception of the slack bus P_{G1}), generator voltages V_{G}, the transformer tap settings (T_i) and reactive power generations of the capacitor bank ($Q_{\mathrm{C}i}$) [133]. These control variables can be expressed as follows:

$$\begin{aligned} &X=\{P_{\mathrm{G}i}: i\in N_{\mathrm{G}}, i\neq 1\}\cup\{V_{\mathrm{G}j}: j\in N_{\mathrm{G}}\} \\ &\cup\{T_k: k\in N_{\mathrm{T}}\}\cup\{Q_{\mathrm{C}l}: l\in N_{\mathrm{C}}\}. \end{aligned} \tag{4.2.2}$$

where N_{G} represents the set of numbers of generator buses; N_{T} represents the set of numbers of transformer branches; and N_{C} represents the set of numbers of possible reactive power source installation bus. For a generator, the real power output indicates the portion of power generated by the plant that averages over a complete cycle of the AC waveform. Meanwhile, reactive power output indicates the portion of power generated by plants returns to the source in each cycle. In the control variables, generator bus voltages indicate the voltage magnitude and angles of the bus embedded with generator. Tap ratio indicates the voltage ratio between the input and output of a transformer. Reactive power generation of var sources measures the reactive power of an AC electric power generator.

Apart from control variables, there are still so-called dependent variables, including generator real power outputs at slack bus $P_{\mathrm{G}1}$, load bus voltages V_{L}, generator reactive power outputs Q_{G} and apparent power flow S. These dependent variables can be denoted as a vector U as follows:

$$\begin{aligned}&U = P_{\mathrm{G}1} \cup \{V_{\mathrm{L}i} : i \in N_{\mathrm{L}}\} \cup \{Q_{\mathrm{G}j} : j \in N_{\mathrm{G}}\}\\ &\cup \{S_k : k \in N_{\mathrm{E}}\}.\end{aligned} \tag{4.2.3}$$

where N_{L} denotes the set of numbers of load buses and N_{E} is the set of numbers of transmission lines in the system. In order to solve the power flow equations in each iteration of the OPF process, the slack bus is chosen by assuming that the slack bus power (including voltage magnitude and voltage phase) is known. For each load bus, both the voltage magnitude and angle are unknown and can be solved. For generator buses, the voltage angle can also be solved. The dependent variables U can be calculated through Newton-Raphson method, by solving the equality constraints, which are formulated as nonlinear power flow equations as follows:

$$\begin{aligned}&P_{\mathrm{G}i} = P_{\mathrm{D}i} + V_i \textstyle\sum_{j \in N_i} V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij})\\ &i \in N_0\\ &Q_{\mathrm{G}i} = Q_{\mathrm{D}i} + V_i \textstyle\sum_{j \in N_i} V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij})\\ &i \in N_{\mathrm{PQ}}\end{aligned} \tag{4.2.4}$$

where $P_{\mathrm{D}i}$ and $Q_{\mathrm{D}i}$ are demanded real and reactive power at bus i respectively; G and B are the real and imaginary part of the admittance matrix of the system respectively; N_i is the set of numbers of buses adjacent to bus i including bus i; N_{PQ} and N_0 are sets of the numbers of PQ buses and total buses (excluding slack bus) respectively. These equality constraints make the input and output real power (or reactive power) equal at each bus.

Additionally, the optimisation problem is subject to some inequality constraints, which are listed as follows:

$$\begin{aligned}
&{P_{\mathrm{G}i}}^{\min} \leq P_{\mathrm{G}i} \leq {P_{\mathrm{G}i}}^{\max} \quad i \in N_{\mathrm{G}} \\
&{Q_{\mathrm{G}i}}^{\min} \leq Q_{\mathrm{G}i} \leq {Q_{\mathrm{G}i}}^{\max} \quad i \in N_{\mathrm{G}} \\
&{V_{\mathrm{G}i}}^{\min} \leq V_{\mathrm{G}i} \leq {V_{\mathrm{G}i}}^{\max} \quad i \in N_{\mathrm{G}} \\
&{V_{\mathrm{L}i}}^{\min} \leq V_{\mathrm{L}i} \leq {V_{\mathrm{L}i}}^{\max} \quad i \in N_{\mathrm{L}} \\
&|S_i| \leq S_i^{\max} \quad i \in N_{\mathrm{E}} \\
&T_i^{\min} \leq T_i \leq T_i^{\max} \quad i \in N_{\mathrm{T}} \\
&{Q_{\mathrm{C}i}}^{\min} \leq Q_{\mathrm{C}i} \leq {Q_{\mathrm{C}i}}^{\max} \quad i \in N_{\mathrm{C}}
\end{aligned} \tag{4.2.5}$$

The constraints of the control variables, $V_{\mathrm{G}i}$ and $Q_{\mathrm{G}i}$, can be satisfied if the search range is defined during the process of optimisation. In addition, the tap position of transformer T and the amount of reactive power source installation Q_{C} are self-constrainted [124]. With the exception of the control variables, the non-linear constrains of other variables can be satisfied using a penalty function, which is commonly used to transform a constrained optimisation problem into an unconstrained one [134]. In this way, maintaining Q_{G}, in slack bus and all PV-buses, is taken into consideration by adding a penalty term, with the inclusion of penalty factor λ_{G}. Then the objective function can be adjusted as:

$$F(X,U) = J + \lambda_{\mathrm{G}} \sum_{i \in N_{\mathrm{G}}^{\lim}} (Q_{\mathrm{G}i} - {Q_{\mathrm{G}i}}^{\lim}) \tag{4.2.6}$$

where

$${Q_{\mathrm{G}i}}^{\lim} = \begin{cases} {Q_{\mathrm{G}i}}^{\max} & \text{if } Q_{\mathrm{G}i} > {Q_{\mathrm{G}i}}^{\max} \\ {Q_{\mathrm{G}i}}^{\min} & \text{if } Q_{\mathrm{G}i} < {Q_{\mathrm{G}i}}^{\min}. \end{cases} \tag{4.2.7}$$

$N_{\mathrm{G}}^{\lim}$ denotes the set of numbers of generator buses on which the injected reactive power is beyond the limit. This way of constraining $Q_{\mathrm{G}i}$ also holds true for limiting $V_{\mathrm{L}i}$, whose constraint is included in the objective function using a penalty factor, λ_{V}.

Objectives: power system dispatch and voltage stability

The problem presented here consists of three cases, and each case has its own objective function:

Case 1: Minimisation of fuel cost. The objective of this task is to minimise the total fuel cost, which is described as:

$$J = \sum_{i=1}^{N_{\mathrm{G}}} \tilde{f}_i \tag{4.2.8}$$

where $\tilde{f}_i$ is the fuel cost (\$/h) of the ith generator, defined as follows:

$$\tilde{f}_i = a_i + b_i P_{\mathrm{G}_i} + c_i P_{\mathrm{G}_i}^2 \tag{4.2.9}$$

where a_i, b_i and c_i are fuel cost coefficients, and P_{G_i} is the real power output generated by the ith generator.

Case 2: Voltage profile improvement. This task aims at minimising fuel cost, while having a flatter voltage profile. The objective function is to minimise the fuel cost, and at the same time to improve voltage profile by minimising the load bus voltage deviations from 1.0 per unit. The objective function can be formulated as:

$$J = \sum_{i=1}^{N_{\mathrm{G}}} \tilde{f}_i + \omega \sum_{i \in N_{\mathrm{L}}} |V_i - 1.0| \tag{4.2.10}$$

where ω is the weighting factor.

Case 3: Voltage stability enhancement. This task is to minimise the fuel cost and enhance voltage stability profile throughout the whole network. Voltage stability assessment is carried out through a global indicator $L_{\max}$, which expresses the risk of a voltage collapse [135]. It is defined as the maximum value of L-index, which is the stability indicator at every bus of the system, as given in the following equation [136][134]:

$$L_{\max} = \max\{L_k, k = 1, \cdots, N_{\mathrm{L}}\} \tag{4.2.11}$$

and L can be calculated from the following equation:

$$L_j = \left|1 + \frac{V_{0j}}{V_{\mathrm{L}_j}}\right| = \left|1 + \frac{S_j^+}{Y_{jj}^{+*} V_{\mathrm{L}_j}^2}\right| \tag{4.2.12}$$

where Y_{jj}^+ is the transformed admittance, $Y_{jj}^+ = 1/Z_{jj}$; S_j^+ is the transformed power $S_j^+ = S_j + S_j^{\mathrm{cor}}$; and

$$S_j^{\mathrm{cor}} = \left[\sum_{i \in N_L} \left(\frac{Z_{ij}^*}{Z_{ji}^*}\right) \cdot \frac{S_i}{V_{\mathrm{L}_i}}\right] \cdot V_{\mathrm{L}_j} \tag{4.2.13}$$

One way of determining L is:

$$L = \max_{j \in N_{\mathrm{L}}} \left| 1 - \frac{\sum_{i \in N_G} F_{ij} \cdot V_i}{V_j} \right| \tag{4.2.14}$$

where V_j is the voltage at load bus j; V_i is the complex voltage at generate bus i; F_{ij} is the element of matrix $[F]$ determined by:

$$[F] = -\frac{[Y_{LL}]}{[Y_{LG}]} \tag{4.2.15}$$

where $[Y_{LL}]$ and $[Y_{LG}]$ are sub-matrices of the Y-bus matrix.

The objective function can be formulated as:

$$J = \sum_{i=1}^{N_{\mathrm{G}}} \tilde{f}_i + \omega L_{\max} \tag{4.2.16}$$

4.2.2 Simulation results

The control variables, including V_{G}, P_{G}, T_i and $Q_{\mathrm{C}i}$, are initialized based on their constraints. Simulation studies are carried out based on the IEEE 30-bus system and IEEE 57-bus system respectively. FOLA is fully compared with improved PSO [45] and GA [16]. In order to have a fair comparison among the algorithms, the publicly available GA and PSO toolboxes are employed in the simulation studies. The GA algorithm, whose code is provided by the genetic algorithm optimisation toolbox [137], is real-coded with heuristic crossover and uniform mutation, and the selection function is normalized geometric ranking [137]. All the control parameters, $e.g.$, mutation rate and crossover rate, $etc.$, are set to default values as recommended in [137]. The code of PSO algorithm is provided by a particle swarm optimisation toolbox for MATLAB [138]. The parameters adopt the default setting in the toolbox: the acceleration factors c_1 and c_2 are both 2.0; and a decaying inertia weight ω starting at 0.9 and ending at 0.4 is used. In order to compare the algorithms fairly, the same N_{femax} is set for FOLA, PSO and GA. For the three algorithms, parameters λ_{V} and λ_{G} are set to 5,100 and 0.1 respectively. The population size used in PSO and GA is set to 48, and the maximal number of generations is limited by parameter N_{femax}. The other parameters setting of PSO and GA follows the suggestions in [60][139]. The parameters setting of FOLA is the same as that given in Section 2.3.2, with the exception of $D = 5$ and N_{femax}.

IEEE 30-bus system

The first test case is concerned with the IEEE 30-bus system, which consists of 48 branches and 6 generators, as shown in Fig. 4.2. Six generator buses are selected as PV-buses and a $V\theta$-bus as follows:

- PV-buses: Bus 2, 5, 8, 11, 13;
- $V\theta$-bus: Bus 1.

The others are PQ-buses (*i.e.* load buses). FOLA, PSO and GA are applied to

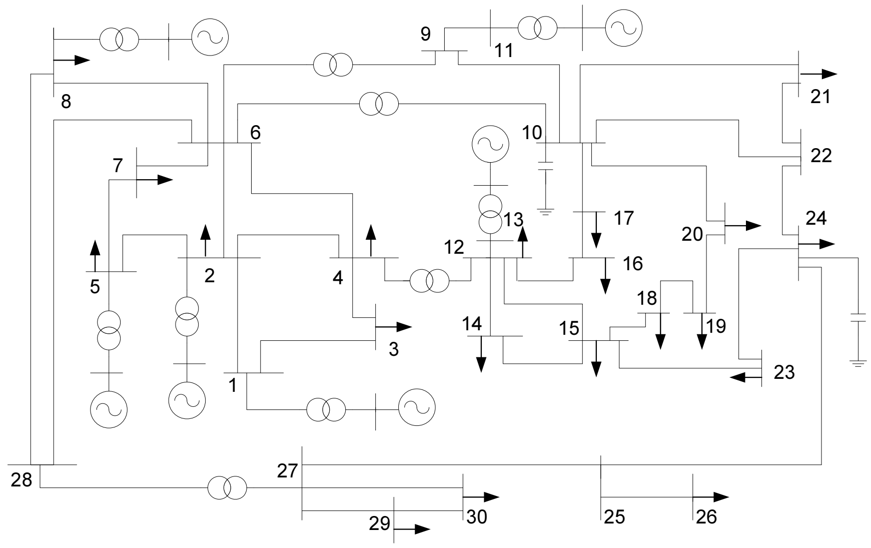


Figure 4.2: Single-line diagram of the IEEE 30-bus system

optimise objective function (4.2.6) by finding control variables X, which includes five generator real power outputs, six generator voltage, four transformer tap settings and nine reactive power generations of the capacitor bank. Parameter N_{femax} used by the three algorithms is set to 1800. The limitation of load voltage of the power system is given as follows: $V_{\mathrm{L}_i}^{\min} = 0.95$; $V_{\mathrm{L}_i}^{\min} = 1.05$. The three algorithms are used to solve the optimisation problems regarding the three cases given in Section 4.2.1.

In case 1, the objective is to minimise the total fuel cost, as given in (4.2.8). The comparison is undertaken among the three algorithms over 30 independent runs, and

Table 4.1: The performance comparison among FOLA, PSO and GA in case 1 on the IEEE 30-bus system

Algorithms	Fuel cost ($/h)	$\sum$ voltage deviations	$L_{\max}$
FOLA	802.1757	1.1908	0.1367
PSO	802.4213	1.1286	0.1370
GA	806.4662	1.0993	0.1371

Table 4.2: The performance comparison among FOLA, PSO and GA in case 2 on the IEEE 30-bus system

Algorithms	Fuel cost ($/h)	$\sum$ voltage deviations
FOLA	804.4814	0.1350
PSO	805.0015	0.1597
GA	807.9653	0.1671

the results are given in Table 4.1. In comparison with PSO, the fuel cost saved by FOLA is 0.2456 ($/h), and the voltage stability of FOLA is enhanced. The sum of voltage deviation of FOLA is larger than that of PSO. This is because the objective of this case only includes the minimisation of fuel cost. It should be mentioned that minimising the fuel cost and enhancing voltage profile are conflicting in this case. FOLA tends to find the solutions which consume lower fuel cost, while keeping the voltages within the range of predefined constraints. In comparison with GA, FOLA saves the fuel cost by 4.2905 ($/h), and is able to enhance the voltage stability.

In case 2, since both fuel cost and voltage deviation are concerned in the objective function, the sum of voltage deviations is reduced by FOLA in this case, in comparison with that in solving case 1. It can be seen from Table 4.2 that FOLA makes a trade-off between the two objectives in this case, compared with its results given in Table 4.1. FOLA outperforms the other two algorithms, as it reduces the fuel cost and decreases the sum of voltage deviations.

In case 3, the objectives include the minimisation of the fuel cost and enhancement of the voltage stability. Table 4.3 lists the experimental results. FOLA performs better than PSO and GA in terms of the fuel cost and voltage stability.

The computation time of the three algorithms is investigated on the IEEE 30-bus system, as given in Table 4.4. It can be seen that FOLA greatly reduces the compu-

Table 4.3: The performance comparison among FOLA, PSO and GA in case 3 on the IEEE 30-bus system

Algorithms	Fuel cost ($/h)	$L_{\max}$
FOLA	803.2916	0.1287
PSO	805.4232	0.1293
GA	805.7327	0.1322

Table 4.4: The computation time (s) consumed by FOLA, PSO and GA on the IEEE 30-bus system

Algorithms	FOLA	PSO	GA
Time (s)	8.4365	12.4635	14.1995

tation time. Compared with PSO, FOLA saves the computation time by 32.31%. In comparison with GA, FOLA saves the computation time by 40.59%.

IEEE 57-bus system

The three algorithms are also evaluated on the IEEE 57-bus system, which consists of 80 branches and 7 generators, as shown in Fig. 4.3. Seven buses are selected as PV-buses and a $V\theta$-bus as follows:

- PV-buses: Bus 2, 3, 6, 8, 9, 12.
- $V\theta$-bus: Bus 1.

The others are PQ-buses. The power demand in the actual system is given as follows: $\sum P_{\mathrm{L}} = 12.508$ p.u.; $\sum Q_{\mathrm{L}} = 3.364$ p.u. The voltage constraint of the system is [0.94 1.06]. The three algorithms are applied to solve the three cases described in Section 4.2.1, by optimising the set of control variables which include six generator real power outputs, seven generator voltage, seventeen transformer tap settings and three reactive power generations of the capacitor bank. N_{femax} is set to 2600 for FOLA, PSO and GA.

FOLA, PSO and GA are applied to solve the three cases on the IEEE 57-bus system, and the experimental results obtained from the three cases are given in Table 4.5, 4.6 and 4.7 respectively. FOLA performs much better than GA in terms of the

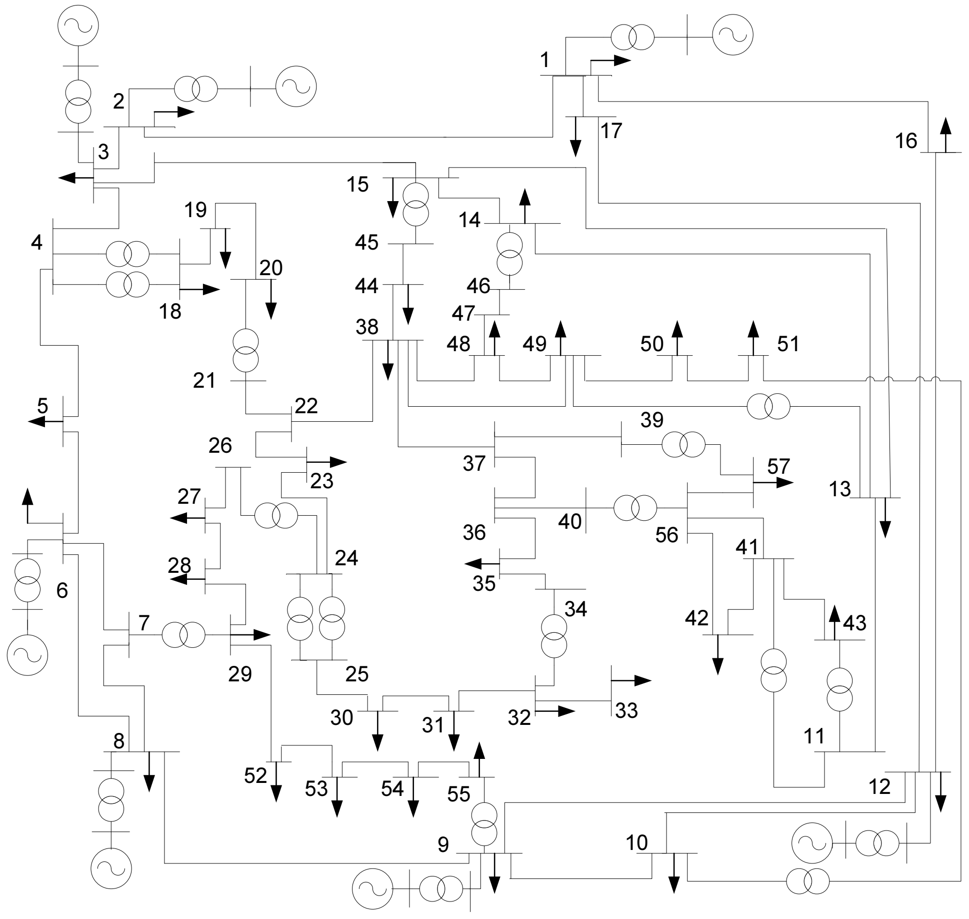


Figure 4.3: Single-line diagram of the IEEE 57-bus system

minimisation of fuel cost. FOLA saves fuel cost by 138.7483 \$/h in case 1, 216.6675 \$/h in case 2 and 484.3370 \$/h in case 3, in comparison with GA. In this simulation study, the voltage profile is improved by reducing the sum of voltage deviation from 0.9520 (GA) to 0.7656 (FOLA), and the voltage stability is greatly enhanced from 0.3597 (GA) to 0.2425 (FOLA). In comparison with PSO, FOLA offers much better performance with respect to cases 2 and 3. The fuel cost saved by FOLA in IEEE 57-bus is much more than that obtained from the IEEE 30-bus system.

In addition, the computation time of the three algorithms is evaluated on the IEEE 57-bus system, and the computation time is listed in Table 4.8. Compared with PSO and GA, FOLA greatly reduces the computation time by saving 38.56% than PSO and 42.10% than GA. It can be seen that the percentage of the computation

Table 4.5: The performance comparison among FOLA, PSO and GA in case 1 on the IEEE 57-bus system

Algorithms	Fuel cost ($/h)	$\sum$ voltage deviations	$L_{\max}$
FOLA	3183.8236	2.8113	0.2480
PSO	3190.7029	2.5768	0.2841
GA	3322.5719	3.3617	0.3547

Table 4.6: The performance comparison among FOLA, PSO and GA in case 2 on the IEEE 57-bus system

Algorithms	Fuel cost ($/h)	$\sum$ voltage deviations
FOLA	3183.6286	0.7656
PSO	3191.1694	0.9828
GA	3400.2961	0.9520

Table 4.7: The performance comparison among FOLA, PSO and GA in case 3 on the IEEE 57-bus system

Algorithms	Fuel cost ($/h)	$L_{\max}$
FOLA	3168.4892	0.2425
PSO	3175.2833	0.3037
GA	3652.8262	0.3597

Table 4.8: The computation time (s) consumed by FOLA, PSO and GA on the IEEE 57-bus system

Algorithms	FOLA	PSO	GA
Time (s)	26.389	42.948	45.580

time saved by FOLA on the IEEE 57-bus system is more than that on the IEEE 30-bus system.

4.3 Evaluation in Dynamic Wind Power Penetrated Systems

4.3.1 Problem formulation

The majority of wind turbine market comprises wind turbines equipped with either fixed-speed or variable-speed converter-controlled induction generators. This section concentrates on fixed-speed wind turbines. When wind power is penetrated into power systems, the characteristics of the wind generators will affect the power flow calculation. Therefore, the model of the wind generators used in the simulation is introduced first in this section.

Wind generator model

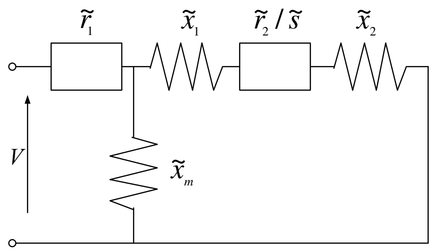


Figure 4.4: Simplified equivalent circuit of asynchronous generator

Asynchronous generators are the most commonly used generators in wind generators. They absorb kinetic energy to produce electrical energy and amount of reactive power to establish their magnetic fields. The simplified equivalent circuit of an asynchronous wind generator is shown in Fig. 4.4, where $\tilde{x}_m$ is the excitation reactance, $\tilde{x}_1$ and $\tilde{x}_2$ are the stator and rotor reactance respectively, $\tilde{r}_2$ is the rotor resistance, $\tilde{r}_1$ is the stator resistance which can be ignored, $\tilde{s}$ is the slip of the asynchronous wind generator, and V_{WG} is wind generator voltage. From this model, the following equations can be obtained [140] [141]:

$$V_{\text{WG}} = \sqrt{\frac{-P_{\text{WG}}(\tilde{s}^2(\tilde{x}_1+\tilde{x}_2)^2+\tilde{r}_2^2)}{\tilde{r}_2\tilde{s}}} \tag{4.3.1}$$

$$Q_{\text{WG}} = -\left(\frac{V_{\text{WG}}^2}{\tilde{x}_m} + \frac{P_{\text{WG}}(\tilde{x}_1+\tilde{x}_2)}{r_2}\tilde{s}\right) \tag{4.3.2}$$

From (4.3.1), we can get

$$\tilde{s} = \frac{\left(-V_{\text{WG}}^2\tilde{r}_2 + \sqrt{V_{\text{WG}}^4\tilde{r}_2^2 - 4P_{\text{WG}}(\tilde{x}_1+\tilde{x}_2)^2\tilde{r}_2^2}\right)}{2P_{\text{WG}}(\tilde{x}_1+\tilde{x}_2)^2} \tag{4.3.3}$$

The Q-V character equation of the asynchronous wind generator can be obtained by integrating (4.3.3) into (4.3.2):

$$Q_{\text{WG}} = f(V_{\text{WG}}) = -\frac{V_{\text{WG}}^2}{\tilde{x}_m} + \frac{-V_{\text{WG}}^2 + \sqrt{V_{\text{WG}}^4 - 4P_{\text{WG}}^2(\tilde{x}_1+\tilde{x}_2)^2}}{2(\tilde{x}_1+\tilde{x}_2)} \tag{4.3.4}$$

where P_{WG} denotes the active power of wind farm. If P_{WG} is constant, according to (4.3.4), reactive power Q_{WG} that is absorbed by asynchronous generator during operation has a close relationship with voltage V_{WG}.

OPF in wind power penetrated systems

The optimisation problem here is concerned with the minimisation of a specific objective function of the power system (4.2.1), by optimising control variables within a set of operational and physical constraints, as introduced in Section 4.2.1. Control variables consist of those given in Section 4.2.1, together with the real power outputs provided by the wind power generators.

When the wind power is penetrated in the power system, the power flow calculation should modify the active and reactive power balance of node incorporated wind farms, $i.e.$ (4.2.4), with the characteristics of wind generators as given in (4.3.4).

If node i is incorporated with wind farm, the maximum value of wind power generated is determined by wind velocities, which can be regarded as a constant in a short period. To model the time-varying wind generator configuration, we use the approximation of multiple time periods, in which the maximum wind power output varies as the time period proceeds. In fact, a wind power output is unknown, which requires an accurate prediction to obtain. This chapter is not concerned with power forecasting and here the wind power output is given assuming it is known. If the active output of the wind turbine is set, its reactive output will be regulated to meet the voltage regulation requirement according to (4.3.4). Therefore, if calculating the power flow of the wind power penetrated system by Newton-Raphson method, the element $\frac{\partial \triangle Q_{\mathrm{WG}}}{\partial V_{\mathrm{WG}}} V_{\mathrm{WG}}$ situating in the Jacobian matrix needs to be modified according to the following equation:

$$\frac{\partial Q_{\mathrm{WG}}}{\partial V_{\mathrm{WG}}} = -\frac{2V_{\mathrm{WG}}}{\tilde{x}_m} + \frac{-V_{\mathrm{WG}}\sqrt{V_{\mathrm{WG}}^4 - 4P_{\mathrm{WG}}^2(\tilde{x}_1+\tilde{x}_2)^2} + V_{\mathrm{WG}}^3}{(\tilde{x}_1+\tilde{x}_2)\sqrt{V_{\mathrm{WG}}^4 - 4P_{\mathrm{WG}}^2(\tilde{x}_1+\tilde{x}_2)^2}} \quad , \tag{4.3.5}$$

while the rest calculation process is the same as the traditional process based on Newton-Raphson method [127] [142].

Objectives

Case 1: The objective of this task is to minimise the total fuel cost of (4.2.8), $i.e.$ $J = \sum_{i=1}^{N_{\mathrm{G}}} \tilde{f}_i$.

Case 2: This task is to minimise fuel cost and enhance voltage stability throughout the whole network, which is carried out through a global indicator $L_{\max}$. J can be formulated as (4.2.16).

4.3.2 Simulation results

Compared with classical EAs

Simulation studies are carried out on the IEEE 30 and 57-bus power systems respectively, which are incorporated with wind generators, whose model is provided in Section 4.3.1. The parameters of the wind generator are listed in Table 4.9. The time-varying configuration of the wind generator is given in Table 4.10, which lists

Table 4.9: The parameters setting of induction generators

Parameter	$\tilde{x}_1$	$\tilde{r}_2$	$\tilde{x}_2$	$\tilde{x}_m$	$\tilde{s}$
Value (p.u.)	0.0100	0.1091	0.0037	3.4547	-0.0040

Table 4.10: The maximum real power output (MW) of the wind generator in multiple time periods

time periods	1	2	3	4	5
IEEE 30-bus	30	50	70	100	60
IEEE 57-bus	200	300	400	500	250

the maximum real power outputs generated in different time periods. FOLA is fully compared with the improved PSO and GA. The parameter settings of PSO and GA are the same as those given in Section 4.2.2. The parameters of FOLA are also the same as those given in Section 4.2.2, with the exception of $D = 15$ and N_{femax}. The same N_{femax} is employed by algorithms FOLA, PSO and GA in the same time period.

- The first test case is concerned with the IEEE 30-bus system, which consists of 48 branches and 6 generators, as shown in Fig. 4.2. In the simulation, the fuel generator locating on the 8th node is replaced with a time-varying wind generator, whose maximum real power output varies in different time periods, as given in Table 4.10. For each time period, the maximum number of function evaluations, N_{femax}, is set to 1,000.

  FOLA, PSO and GA are applied to minimise the objective function, (4.2.6), by optimising the control variables, which consist of five generator real power outputs (including the fuel generators and wind generator), six generator voltages, four transformer tap settings and ten reactive power generations of the capacitor bank. The three algorithms are used to solve the two cases of problems given in Section 4.3.1.

  In case 1, the objective is to minimise the total fuel cost, as given in (4.2.8). The performance of FOLA, PSO and GA can be observed in Fig. 4.5. FOLA is able to track the changes of the power system configuration more rapidly

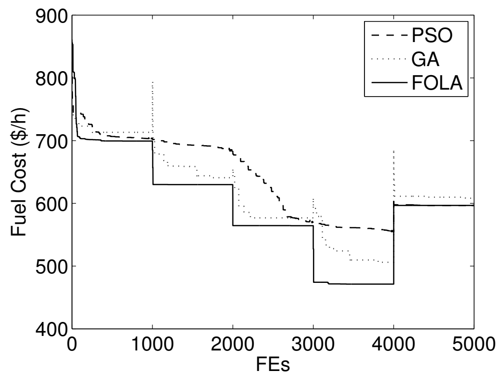


Figure 4.5: The performance comparison among PSO, GA and FOLA in solving case 1 on the modified IEEE 30-bus system

and accurately than the other algorithms. As for PSO, it fails to respond to the changes most of the time, with a constantly decreasing performance in time periods 2-4. It is difficult for PSO to respond to this changing frequency, as the maximum number of FEs constrained in each time period is not enough for PSO to converge to the global minimum before the next time period comes. Additionally, the convergence speed of PSO obtained in time period 2 is much slower than that in time period 1. It is partly because PSO was developed delicately for static optimisation problems. In the case of static optimisation, the global optimum found in each generation is used as a selection reference to reproduce individuals of the next generation. However, in a dynamic power system, the information of global optimum may not be necessarily useful to guide the selection, as the real optimum of the environment varies from time to time. Thus, the information obtained from a generation is of limited use for the next generation in the dynamic environment. Furthermore, the information of the previous generation could even hamper the search when the system changes, due to PSO clings on to the previous best solution in despite of the changing configuration of the system. In this sense, due to the lack of appropriate diversity, the performance of PSO is not acceptable in this case. Although GA converges faster than PSO, its solution is much worse than that of FOLA in terms of accuracy. The gap between the fuel costs consumed by FOLA and GA indicates that a large amount of energy can be saved if using FOLA rather than GA. In addition, once the power system configuration changes, GA produces a large fluctuation regarding the fuel cost, which to some extent implies the instability of the power system if GA is applied.

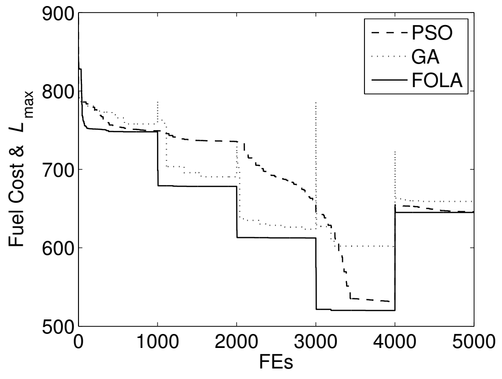


Figure 4.6: The performance comparison among PSO, GA and FOLA in solving case 2 on the modified IEEE 30-bus system

In case 2, the objective is to minimise the fuel cost and to enhance the voltage stability, as described in (4.2.16). The performance of the three algorithms has been presented by the three curves in Fig. 4.6. It can be observed from Fig. 4.6 that the tracking ability of FOLA is constantly better than that of PSO and GA as the function evaluation proceeds. PSO is not able to react to most changes in time. The convergence speed of PSO is much slower than the power system's changing rate, as it cannot converge to the global minimum before the next change occurs. As for GA, it is unable to converge to accurate solutions. In this case, the turbulence of the objective value, caused by GA at the changing moment, is larger than that in case I, due to the voltage stability is also concerned in the objective. This fact reveals that GA could damage the system if the configuration of the power system changes frequently.

The computation time of these three algorithms is investigated on the modified IEEE 30-bus power system, as given in Table 4.11, which lists the average computation time consumed by the three algorithms in one time period. It can be seen that FOLA greatly reduces the computation time. Compared with PSO, FOLA saves the computation time by 31.17%. In comparison with GA,

Table 4.11: The computation time (s) consumed by FOLA, PSO and GA on the IEEE 30-bus system

Algorithms	FOLA	PSO	GA
Time (s)	8.685	12.618	13.754

FOLA saves the computation time by 36.85%.

- The three algorithms are also evaluated on the IEEE 57-bus system, which consists of 80 branches and 7 generators, as shown in Fig. 4.3. To integrate wind power in the power system, the fuel generator located at the 8th node is replaced with a wind generator whose configuration varies in different time periods, as provided in Table 4.10. In each time period, the maximum number of FEs is set to 1,500. The three algorithms are applied to solve the two cases given in Section 4.3.1, by optimising the control variables which include six generator real power outputs (including the wind generator), seven generator voltage, seventeen transformer tap settings and four reactive power generations of the capacitor bank.

  FOLA, PSO and GA are applied to solve the problem on the modified IEEE 57-bus system. The tracking ability of the three algorithms is presented in Figs. 4.7 and 4.8, which are the results corresponding to cases 1 and 2 respectively. FOLA is able to track the global minimum and constantly performs better than the other two algorithms. The curve obtained by FOLA reveals that the variation of its objective value is in line with the variation of the system configuration, which is applied on the wind generator. On the modified IEEE 57-bus system, GA completely fails to solve the problems, due to the complexity of the problems. The performance of PSO on this system is better than that on IEEE 30-bus system, partly because more FEs are allowed to be used in one time period. However, compared with FOLA, PSO responds to the changing system configuration with a serious delay, thus cannot obtain a satisfactory tracing performance. Between Figs. 4.7 and 4.8, PSO performs worse in the later case, which implies that the performance of PSO deteriorates once the voltage stability is involved in the objective.

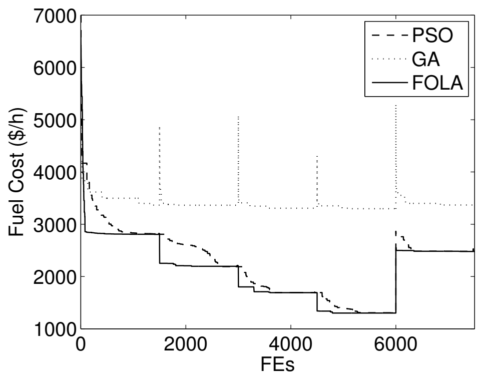


Figure 4.7: The performance comparison among PSO, GA and FOLA in solving case 1 on the modified IEEE 57-bus system

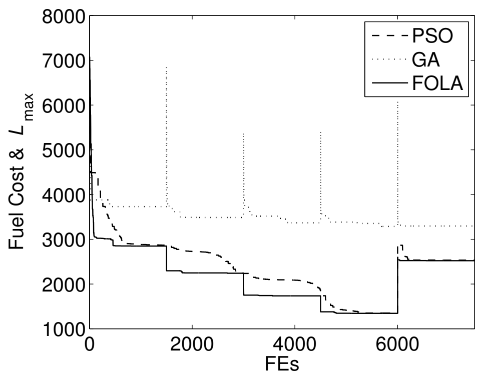


Figure 4.8: The performance comparison among PSO, GA and FOLA in solving case 2 on the modified IEEE 57-bus system

In addition, the computation time of these three algorithms is evaluated on the modified IEEE 57-bus system, and their computation time is listed in Table 4.12. Comparing with PSO and GA, FOLA greatly reduces the computation time by saving 34.39% and 37.78% respectively. It can be seen that the percentage of computation time saved by FOLA on the modified IEEE 57-bus system is larger than that on the modified IEEE 30-bus system.

Compared with recently-proposed EAs

FOLA has also been compared with Cooperative Particle Swarm Optimisation (CPSO) [50] and Comprehensive Learning Particle Swarm Optimisation (CLPSO)

Table 4.12: The computation time (s) consumed by FOLA, PSO and GA on the modified IEEE 57-bus system

Algorithms	FOLA	PSO	GA
Time (s)	24.055	36.666	38.664

[51], in solving the optimal power flow problems in wind power penetrated power systems. The test case here is concerned with the IEEE 118-bus power system, in which there are 54 fuel generators, 118 buses, 186 branches, 91 load sides and 54 thermal units, as given in Fig. 4.9. Based on the IEEE 118-bus system, the fuel generator locating at bus 8 is replaced by wind generators, whose model is provided in Section 4.3.1. The parameters of the wind generator are the same as those given in Table 4.9. In the simulation, the whole optimisation process is divided into five time periods, in which the wind generators provide different maximum real power outputs. For the five time periods, the maximum real power outputs of the wind generator are set to 200, 500, 800, 1000, 300 MW respectively. In each time period, N_{femax} is set to 6,000 for each algorithm. The problem concerned here has 130 control variables to be optimised, including real power outputs of both the fuel generators and wind generators, generator voltage, transformer tap settings and reactive power generations of capacitor banks. In other words, the dimensionality of the problem is 130.

Each function evaluation involves one power flow calculation, which consumes much more computation time than the algorithm itself. Therefore the number of power flow calculation is very precious during the optimisation process. Within a limited numbers of FEs, the efficiency and convergence rate are of great significance for the optimal power flow problem, especially on large-scale complex power systems. The convergence characteristics obtained by FOLA, CLPSO and CPSO in solving cases 1 and 2 can be observed in Fig. 4.10 (a) and (b) respectively.

It can be seen that FOLA is able to track the changes of the power system configuration more rapidly and accurately than the other two algorithms. At the beginning of time period 5, $i.e.$ when FEs= 2.4×10^4, the fuel cost obtained by FOLA increases compared to that in time period 4, due to the real power outputs provided by the wind generators decrease in period 5, and fuel generators have to increase

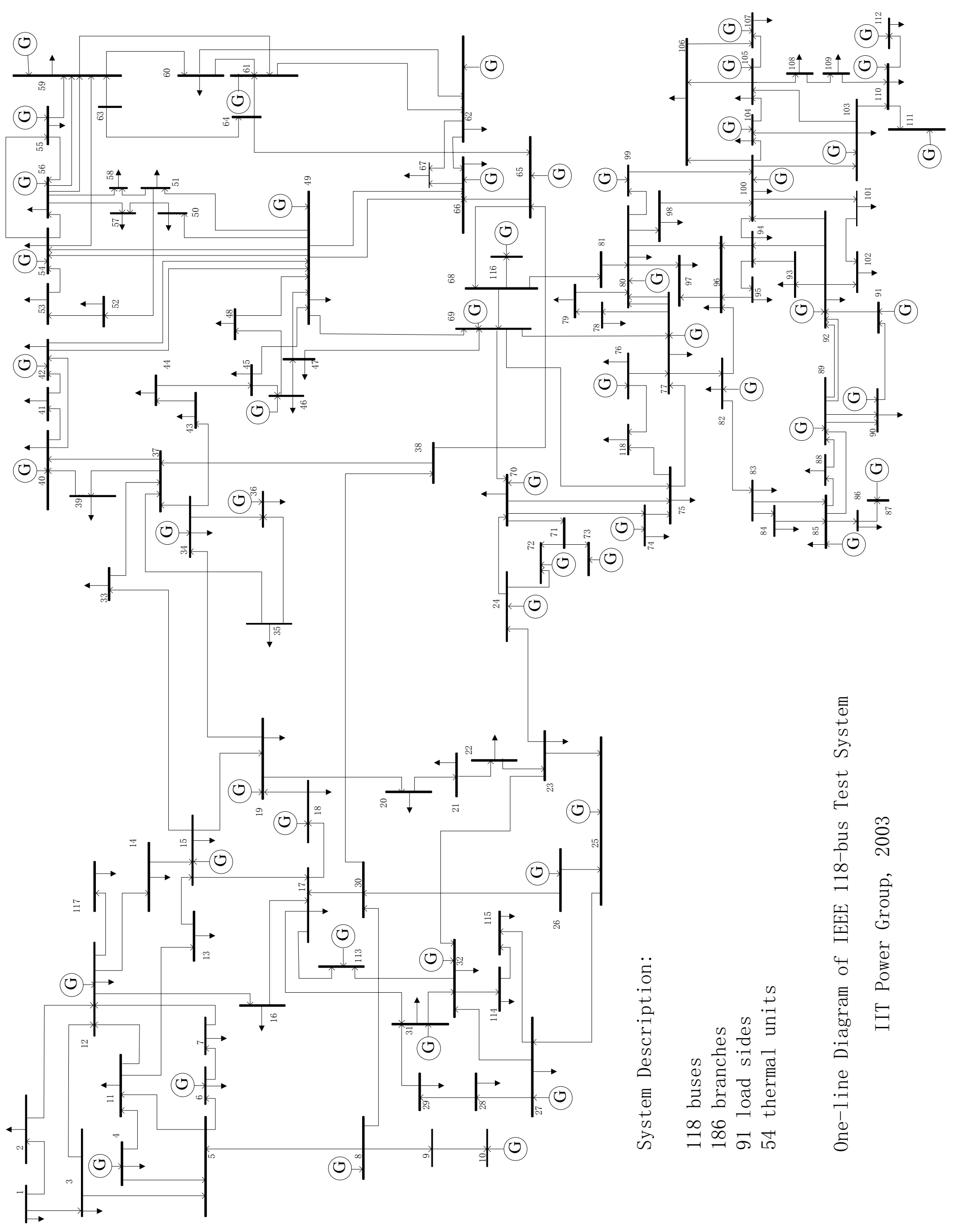
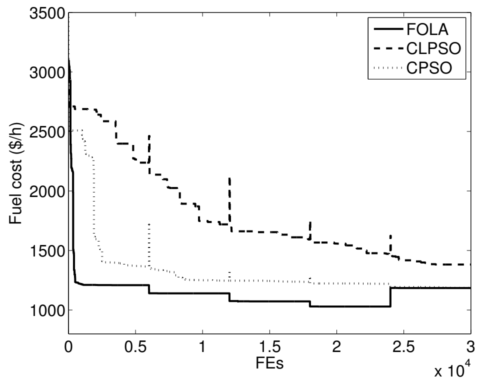
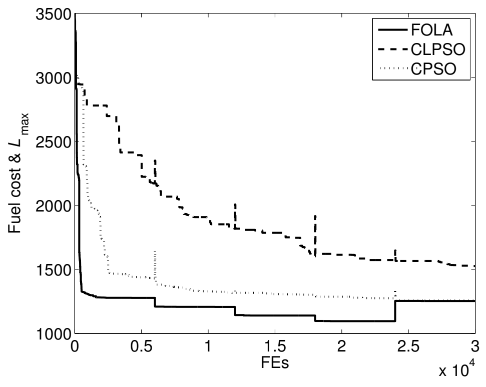


Figure 4.9: Single-line diagram of the IEEE 118-bus system

the real power outputs and complement this gap. The convergence speed of PSO is much slower than the changing rate of the system configuration. It can be seen that the maximum number of FEs constrained in each time period is not enough for CLPSO to converge to the global minimum before the next time period comes. As for CPSO, the fuel cost obtained by CPSO almost remains the same after 6,000 FEs, even when the configuration of the power system has been changed. Therefore, CPSO fails to respond to the changes of the system configuration and is trapped in a local optimum. However, CPSO performs better than CLPSO in terms of the convergence rate.



(a)



(b)

Figure 4.10: The performance comparison among FOLA, CLPSO and CPSO on the IEEE 118-bus power system: (a) for case 1; (b) for case 2

In addition, the minimum fitness values obtained by FOLA, CLPSO and CPSO at each time period are provided in Table 4.13. It can be seen that FOLA can obtain smaller fitness values than CLPSO and CPSO constantly.

The computation time consumed by the algorithms is also investigated in this case. The computation time concerned here does not include the time used for power flow calculations. Table 4.14 lists the computation time consumed by FOLA, CLPSO and CPSO. It can be seen that FOLA consumes less computation time than CLPSO and CPSO.

4.4 Conclusions

In this chapter, FOLA has been used to solve optimal power flow problems. The problems are concerned with three cases, in which the objective functions are the

Table 4.13: The minimum fitness values obtained by FOLA, CLPSO and CPSO in different time periods

		FOLA	CLPSO	CPSO
Case 1	Minimum in period 1	1.2085e+003	2.2379e+003	1.3653e+003
	Minimum in period 2	1.1396e+003	1.7174e+003	1.2464e+003
	Minimum in period 3	1.0727e+003	1.5921e+003	1.2367e+003
	Minimum in period 4	1.0290e+003	1.4620e+003	1.2201e+003
	Minimum in period 5	1.1849e+003	1.3825e+003	1.1858e+003
		FOLA	CLPSO	CPSO
Case 2	Minimum in period 1	1.2776e+003	2.1717e+003	1.4295e+003
	Minimum in period 2	1.2072e+003	1.8179e+003	1.3170e+003
	Minimum in period 3	1.1400e+003	1.6019e+003	1.3019e+003
	Minimum in period 4	1.0960e+003	1.5730e+003	1.2749e+003
	Minimum in period 5	1.2530e+003	1.5262e+003	1.2578e+003

Table 4.14: Computation time (s) consumed by FOLA, CLPSO and CPSO

		FOLA	CLPSO	CPSO
Computation time	Case 1	100.55	121.995	131.21
	Case 2	118.83	149.275	169.125

combination of the following objectives: minimisation of the fuel cost, improvement of the voltage profile, and enhancement of the voltage stability. Simulation studies have been carried out on the IEEE 30-bus and 57-bus systems respectively. The simulation results have demonstrated that FOLA is able to offer more accurate solutions with shorter computation times, in comparison with the improved PSO and GA, particularly on the IEEE 57-bus system.

FOLA is also applied to solve the optimal power flow problems in the dynamic wind power penetrated systems. Simulation studies have been carried out among FOLA, the improved PSO and GA, based on the modified IEEE 30-bus and 57-

bus systems, which are embedded with time-varying wind power generators. The simulation results have demonstrated that FOLA is able to track the changes of the power system configuration more rapidly and accurately than the improved PSO and GA, particularly when voltage stability is involved in the objective function. Besides, FOLA is able to offer more accurate solutions with shorter computation time, in comparison with PSO and GA. FOLA is also compared with recently-proposed EAs, CLPSO and CPSO, based on IEEE 118-bus system. Advantages of FOLA have been demonstrated by the fact that FOLA reduces the fuel cost greatly and enhances the voltage stability of the power system.

Chapter 5

The Application of MOLA in Multi-objective Optimal Power Flow Problems

5.0.1 Introduction

In wind power penetrated systems, multi-objective problems have attracted increasing attention in the last decade. Rather than focusing on one objective, the objectives can include the combination of different aspects, such as emission reduction [143], energy loss minimisation [144], power loss minimisation in the transmission lines [127] and fuel cost reduction [129], and so on. In the first two applications of this chapter, two objectives are concerned, and both of them are important to system operation in wind power penetrated systems: (1) The first objective is to reduce the fuel cost and pollutant emission, which is widely known as economic emission dispatch problem [145][143]. This objective is important due to economical and environmental reasons. (2) The second objective is to enhance voltage stability, as voltage instability is a major power system weakness resulting in severe detriments with economical, technical and social dimensions [116][146], as aforementioned in Section 4.2.1. This goal can be achieved by properly distributing real power outputs, and reasonably adjusting other control variables, including wind generator parameters, generator bus voltage magnitudes, transformer tap settings and reactive power of capacitor banks. The merits of MOLA have been demonstrated, in comparison with NSGA-II and MOEA/D. The study is undertaken on the modified IEEE 30-bus power system and new England test power system, which are incorporated with fixed-speed and variable-speed wind generators respectively. MOLA is applied to reduce the operational cost and enhance the system stability simultaneously. The simulation results have shown that: MOLA is superior over MOEA/D with respect to finding the range of the Pareto fronts; MOLA outperforms NSGA-II in terms of the accuracy of the non-dominated solutions.

In the chapter, MOLA is also applied in deregulated electricity market, which creates more competition and more trading mechanisms for market players. However, the emergence of deregulated electricity markets poses new challenges to the solution of the optimal power flow problem. In deregulated market, optimal power flow computation is part of the core pricing mechanism for electricity trading, where energy can be traded in bids and offers. In order to meet their legal obligations of providing timely market settlements and to ensure market fairness and efficiency, the Independent System Operator (ISO) is responsible for the provision of additional services that are necessary to support the transmission of electrical energy while maintaining secure and reliable operation of the power system. ISO dispatches according to the bid price of each player instead of the production cost of each generating unit. Therefore, the conventional objective of optimal power flow has been evolved to be price-based, which is to maximise profits of the spot market and achieve the resource scheduling. This problem has been investigated intensively after the emergence of deregulated market [147] [148], in which the profit is defined as customer benefit minus generator cost. Due to the increasing attention of environmental pollution issue [143], reducing emission pollutants is considered important in the deregulated market [149]. Therefore, the cost of emission is integrated into the social profit in this section, thus the social profit is defined as customer benefit minus generator cost and emission cost. Apart from maximising the social profit, the enhancement of voltage stability is also concerned in the section, as voltage instability is a major power system weakness resulting in severe detriments with economical, technical and social dimensions [116][146].

5.1 Evaluation on Power Systems with Fixed-speed Wind Generators

5.1.1 Problem formulation

The problem presented here is to reduce the operational cost and enhance the voltage stability simultaneously, through optimising control variables within a set of operational and physical constraints, which can be formulated as follows:

$$\begin{aligned} &\min F(X,U) = [f_1(X,U), f_2(X,U)] \\ &\text{s.t.}\ \ g(X,U) = 0;\ h(X,U) \leq 0. \end{aligned} \tag{5.1.1}$$

where f_1 and f_2 are the two objective functions; g and h denote constraint functions [150], as introduced in Section 4.2.1. The set of control variables, X, includes real power outputs (with the exception of the slack bus), generator voltages, the transformer tap settings and reactive power generations of the capacitor bank.

The two objective functions of (5.1.1), f_1 and f_2, are introduced respectively as follows:

- The operational cost (f_1) takes economical and environmental concerns into consideration. It is the sum of two parts: the total fuel cost and the cost of pollutant emission [143]:

  $$f_1 = \sum_{i=1}^{N_{\mathrm{FG}}} \tilde{f}_i + \sum_{i=1}^{N_{\mathrm{FG}}} E_i \tag{5.1.2}$$

  where N_{FG} represents the set of numbers of buses incorporated with fuel generators; $\tilde{f}_i$ and E_i are the fuel cost (\$/h) and the cost of pollutant emission of the ith generator respectively. $\tilde{f}_i$ can be formulated as (4.2.9). E_i can be described as follows:

  $$E_i = h * (10^{-2}(\alpha_{\mathrm{E}i} + \beta_{\mathrm{E}i} P_{\mathrm{G}_i} + \gamma_{\mathrm{E}i} P_{\mathrm{G}_i}^2) + \zeta_{\mathrm{E}i} \exp(\lambda_{\mathrm{E}i} P_{\mathrm{G}_i})) \tag{5.1.3}$$

  where h is price penalty factor [151] for the emission of atmospheric pollutants; $\alpha_{\mathrm{E}i}$, $\beta_{\mathrm{E}i}$, $\gamma_{\mathrm{E}i}$, $\zeta_{\mathrm{E}i}$ and $\lambda_{\mathrm{E}i}$ are the coefficients of the ith generator's emission characteristics.

- Voltage stability assessment is carried out through a global indicator $L_{\max}$, which expresses the risk of a voltage collapse [135]. The indicator can be defined as equation (4.2.11), as given in Section 4.2.1.

5.1.2 Simulation studies

MOLA is fully compared with MOEA/D and NSGA-II, by solving multi-objective optimisation problems in the modified IEEE 30-bus power system and new England test system respectively, which are incorporated with fixed speed wind generators, whose model is given in Section 4.3.1. The parameter settings of MOEA/D and NSGA-II follow the suggestions in [103] and [29] respectively. The parameter settings of MOLA, except for w_{c} and N_{femax}, are the same as previous settings. In addition, the same N_{femax} is used by the three algorithms when solving the same test case. The parameter settings of the fixed-speed wind generators are listed in Table 4.9, and the coefficients in (5.1.3) follow the suggestions given in [143].

Modified IEEE 30-bus power system

The first test case is IEEE 30-bus power system, which consists of 41 branches and 8 generators, as shown in Fig. 4.2. Nodes 21 and 30 are embedded with wind power generators. Control variables X include generator real power outputs, generator voltages, transformer tap settings and reactive power generations of the capacitor bank. N_{femax} is set to 30,000 for MOLA, and 40,000 for both NAGA-II and MOEA/D.

The Pareto fronts obtained by the three algorithms are presented in Fig. 5.1. It is obvious that MOLA outperforms NSGA-II in terms of the accuracy of the obtained solutions and the range of the Pareto fronts. Between MOEA/D and MOLA, their non-dominated solutions overlap in the objective space. However, the range of their Pareto fonts varies widely. To see clearly, the results of the two algorithms are plotted in separate subfigures, as shown in Fig 5.2. The non-dominated solutions found by MOEA/D flock to one end, locating at the range of [609.6, 612.4]$\times$[0.7653, 0.7758]. Among the solutions found by MOEA/D, there is an isolated point, *i.e.*

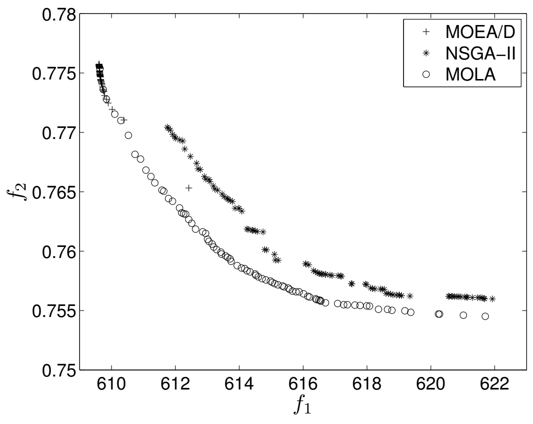


Figure 5.1: Pareto fronts obtained by MOEA/D, NSGA-II and MOLA on the IEEE 30-bus wind power penetrated system

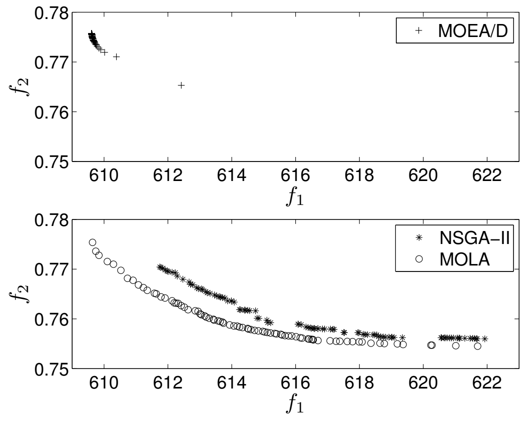


Figure 5.2: Pareto fronts (in separate subfigures) obtained by MOEA/D, NSGA-II and MOLA on the IEEE 30-bus wind power penetrated system

(612.4, 0.7653), and a big gap exists between solutions (612.4, 0.7653) and (610.4, 0.7710), which shows that MOEA/D has difficulty in finding the solutions between the two points. As for MOLA, its non-dominated solutions spread over the range of [609.6, 621.7]$\times$[0.7545, 0.7755], which is wider than that of MOEA/D. In addition, the isolated solution obtained by MOEA/D, (612.4, 0.7653), is dominated by the solutions found by MOLA. MOLA can find more non-dominated solutions (224) than MOEA/D (101). The fact that MOLA finds wider Pareto front and more non-dominated solutions than MOEA/D suggests that MOLA can provide more various choice among the non-dominated solutions that satisfy the optimisation targets.

Table 5.1: The setting of reference solution and extreme solutions; HV and Δ obtained by MOEA/D, NSGA-II and MOLA on the modified IEEE 30-bus power system (including mean and standard deviation)

Settings	Reference solution	Extreme solution 1	Extreme solution 2		
	[622.0000, 0.7770]	[609.5526, 0.7769]	[621.7029, 0.7545]		
Results		HV		Δ	
		Mean	Std	Mean	Std
	MOEA/D	0.1274	0.0045	0.7687	0.3132
	NSGA-II	0.1765	0.0076	0.1918	0.1499
	MOLA	0.2202	0.0028	0.0096	0.0047

The first two rows of Table 5.1 list the reference solution and extreme solutions that are employed to calculate hypervolume HV and diversity metric Δ respectively. The mean and standard deviation of HV and Δ obtained by these algorithms over 20 independent runs are also listed in Table 5.1. It can be seen that MOLA outperforms MOEA/D and NSGA-II, as it finds larger hypevolume and smaller diversity metric on average. In addition, the standard deviation of HV and Δ obtained by MOLA is consistently smaller than that obtained by MOEA/D and NSGA-II. This fact presents the reliability of MOLA in solving the problem of this case.

Modified new England test system

The second test case is a reduced model of the power system in new England, which is modified and incorporated with wind power generators, as given in Fig. 5.3, which consists of 46 branches and 12 generators. Nodes 4 and 8 are incorporated with wind power generators. N_{femax} is set to 40,000 for MOEA/D, NSGA-II and MOLA.

As shown in Fig. 5.4, MOLA greatly presents its superiority over MOEA/D and NSGA-II with respect to the following facts: the non-dominated solutions found by MOLA are much smaller than those found by MOEA/D and NSGA-II; the range of the Pareto front found by MOLA is much wider than that of MOEA/D and NSGA-II. It should be mentioned that MOEA/D and NSGA-II also find other solutions when

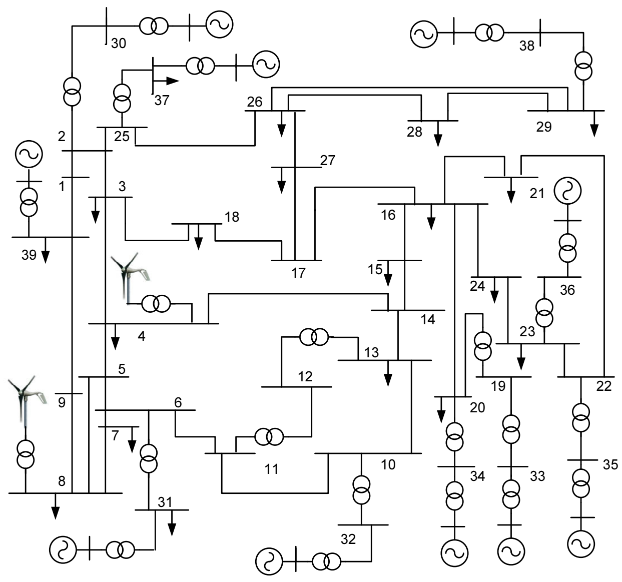


Figure 5.3: Single-line diagram of the modified new England wind power penetrated system

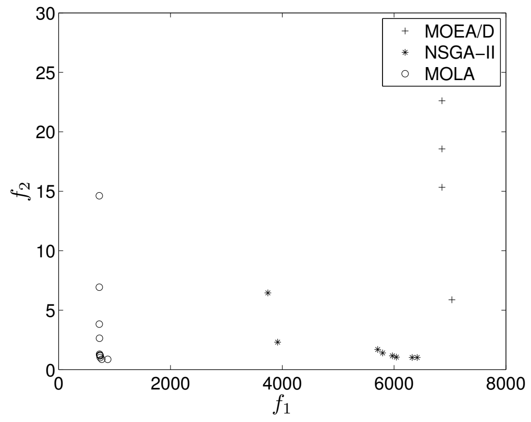


Figure 5.4: Pareto fronts obtained by MOEA/D, NSGA-II and MOLA on the modified new England wind power penetrated system

$f_1 > 8,000$ and $f_2 > 30$, however, they are much worse than the solutions given in the figure, and all of them are dominated by the solutions found by MOLA. They are far beyond the range given in this figure, thus they are not plotted, considering the readability of the figure.

Table 5.2 lists the reference solution, two extreme solutions, and the values of HV and Δ obtained by these algorithms (including mean and standard deviation

Table 5.2: The setting of reference solution and extreme solutions; HV and Δ obtained by MOEA/D, NSGA-II and MOLA on the modified new England test system (including mean and standard deviation)

Settings	Reference solution	Extreme solution 1		Extreme solution 2	
	[8000, 25]	[725.7176, 3.7490]		[1964.3, 0.4110]	
Results		HV		Δ	
		Mean	Std	Mean	Std
	MOEA/D	4.5611×10^4	2.0570×10^4	0.9462	0.2584
	NSGA-II	1.1661×10^5	2.5979×10^4	0.7035	0.1217
	MOLA	1.7718×10^5	1.6797×10^4	0.5687	0.1186

Table 5.3: Load demand (MW) at bus 10

Periods	Hours 1-4	Hours 5-8	Hours 9-12
Load demand	2.5	3.5	5.5
Periods	Hours 13-16	Hours 17-20	Hours 21-24
Load demand	4.7	5	3.8

over 20 independent runs). The results have demonstrated the superiority of MOLA over MOEA/D and NSGA-II. MOLA finds larger HV and smaller Δ than MOEA/D and NSGA-II on average. The standard deviation of HV and Δ obtained by MOLA is consistently smaller than that obtained by MOEA/D and NSGA-II.

Variation of operation condition

The environment constraint concerned in this chapter considers more aspects of variation than that given in Section 4.3.1. Here, the operation condition takes the fluctuation of power demand and wind speed throughout the day into account.

The demand for electricity varies throughout the day. In order to simulate the variation of power demand, a day is divided into six 4-hour periods. Each period has a different power demand, as given in Table 5.3.

When a node is incorporated with the wind generator, the maximum wind power output is determined by wind speed, which varies in different time periods, as given

Table 5.4: Wind speed

Periods	Hours 1-4	Hours 5-8	Hours 9-12
Wind speed	6.0832	7.6643	8.7735
Periods	Hours 13-16	Hours 17-20	Hours 21-24
Wind speed	8.2561	8.7735	7.6643

Table 5.5: The minimum objective fitness values derived by MOEA/D, NSGA-II and MOLA in the dynamic environment

	Hours 1-4		Hours 5-8		Hours 9-12	
	Min f_1	Min f_2	Min f_1	Min f_2	Min f_1	Min f_2
MOEA/D	599.60	0.772	602.43	0.766	608.66	0.766
NSGA-II	603.10	0.758	605.09	0.756	612.20	0.754
MOLA	599.61	0.756	602.42	0.754	608.66	0.753
	Hours 13-16		Hours 17-20		Hours 21-24	
	Min f_1	Min f_2	Min f_1	Min f_2	Min f_1	Min f_2
MOEA/D	606.15	0.765	606.96	0.765	603.47	0.765
NSGA-II	609.71	0.754	611.19	0.754	608.01	0.754
MOLA	606.13	0.753	606.95	0.753	603.45	0.752

in Table 5.4. The mechanical power extracted by a wind turbine from the wind can be expressed by the well-known cube law equation [152]:

$$P_{\mathrm{w}} = \frac{1}{2}\rho_{\mathrm{air}} A_{\mathrm{blade}} C_p v_{\mathrm{w}}^3 \tag{5.1.4}$$

where ρ is the air density; A_{blade}, defined as $\pi \times R_{\mathrm{blade}}^2$ (R_{blade} is the radius of turbine rotor blades), is the area swept by the turbine blades; C_p is turbine performance coefficient; and v_{w} denotes wind speed. The mechanical power extracted from the wind is transferred to electricity power and integrated into the power system through the wind generator model given in Section 4.3.1.

NSGA-II, MOEA/D and MOLA are applied to solve the multi-objective problem in the modified IEEE 30-bus power system which takes time-varying load demand and wind speed into consideration. The minimal objective fitness values obtained by these algorithms in different periods are given in Table 5.5. It can be seen that MOLA outperforms MOEA/D and NSGA-II in different situations. It can find smaller operational cost and smaller voltage stability, with the exception of the first period (hours 1-4), in which MOEA/D finds smaller operational cost than MOLA.

To fully compare the performance of the three algorithms in dynamic environment, the obtained values of HV and Δ (including mean and standard deviation) are listed in Tables 5.6 and 5.7, as well as the settings of reference solution and extreme solutions in different periods. The superiority of MOLA has been confirmed by the fact that MOLA finds larger HV and smaller Δ than MOEA/D and NSGA-II on average.

5.2 Evaluation on Power Systems with Variable-speed Wind Generators

5.2.1 Problem formulation

The objectives of the problem are the same as that given in Section 5.1.1. However, different type of wind generators ($i.e.$ variable-speed wind generators) is concerned in this section. The power output of a wind turbine is determined by the wind speed at the location where it is installed. A relationship between the power generated and the wind speed was proposed in the research of [153]. The active power output of a wind turbine can be formulated as:

$$P_{\text{wt}} = \begin{cases} 0 & 0 \leq v < v_{\text{ci}} \\ a + bv^3 & v_{\text{ci}} < v < v_{\text{ra}} \\ P_{\text{ra}} & v_{\text{ra}} < v < v_{\text{co}} \\ 0 & v > v_{\text{co}} \end{cases} \tag{5.2.1}$$

(5.2.2)

$$a = \frac{P_{\text{ra}} v_{\text{ci}}^3}{v_{\text{ci}}^3 - v_{\text{ra}}^3} \tag{5.2.3}$$

$$b = \frac{P_{\text{ra}}}{v_{\text{ra}}^3 - v_{\text{ci}}^3} \tag{5.2.4}$$

Table 5.6: The setting of reference solution and extreme solutions; HV and Δ obtained by MOEA/D, NSGA-II and MOLA in dynamic environment: Hours 1-12 (including mean and standard deviation)

Hours 1-4	Settings	Reference solution	Extreme solution 1		Extreme solution 2	
		[614 0.777]	[599.609 0.7769]		[613.9797 0.7555]	
	Results		HV		Δ	
			Mean	Std	Mean	Std
		MOEA/D	0.1446	0.0143	0.8296	0.1859
		NSGA-II	0.2034	0.0097	0.1172	0.1412
		MOLA	0.2644	0.0058	0.0944	0.0695
Hours 5-8	Settings	Reference solution	Extreme solution 1		Extreme solution 2	
		[616 0.777]	[602.3748 0.7769]		[615.6365 0.7536]	
	Results		HV		Δ	
			Mean	Std	Mean	Std
		MOEA/D	0.1448	0.0056	0.7643	0.1661
		NSGA-II	0.1928	0.0071	0.1041	0.1432
		MOLA	0.2502	0.0053	0.0062	0.0031
Hours 9-12	Settings	Reference solution	Extreme solution 1		Extreme solution 2	
		[622 0.777]	[608.569 0.7769]		[621.4634 0.7527]	
	Results		HV		Δ	
			Mean	Std	Mean	Std
		MOEA/D	0.1416	0.0098	0.7091	0.3081
		NSGA-II	0.1849	0.0129	0.2520	0.1513
		MOLA	0.2472	0.0031	0.0055	0.0064

Table 5.7: The setting of reference solution and extreme solutions; HV and Δ obtained by MOEA/D, NSGA-II and MOLA in dynamic environment: Hours 13-24 (including mean and standard deviation)

Hours 13-16	Settings	Reference solution	Extreme solution 1		Extreme solution 2	
		[620 0.777]	[606.0995 0.7769]		[619.7494 0.7528]	
	Results		HV		Δ	
			Mean	Std	Mean	Std
		MOEA/D	0.1348	0.0089	0.8305	0.3864
		NSGA-II	0.2019	0.0130	0.1186	0.1305
		MOLA	0.2504	0.0024	0.0853	0.0598
Hours 17-20	Settings	Reference solution	Extreme solution 1		Extreme solution 2	
		[620 0.778]	[606.9328 0.7778]		[619.8721 0.7526]	
	Results		HV		Δ	
			Mean	Std	Mean	Std
		MOEA/D	0.1355	0.0091	0.7745	0.3670
		NSGA-II	0.1900	0.0120	0.1222	0.1380
		MOLA	0.2439	0.0054	0.0554	0.0365
Hours 21-24	Settings	Reference solution	Extreme solution 1		Extreme solution 2	
		[620 0.777]	[603.3384 0.777]		[619.6417 0.7523]	
	Results		HV		Δ	
			Mean	Std	Mean	Std
		MOEA/D	0.1615	0.0213	0.8897	0.2949
		NSGA-II	0.2407	0.0219	0.1199	0.1450
		MOLA	0.3019	0.0131	0.0921	0.1091

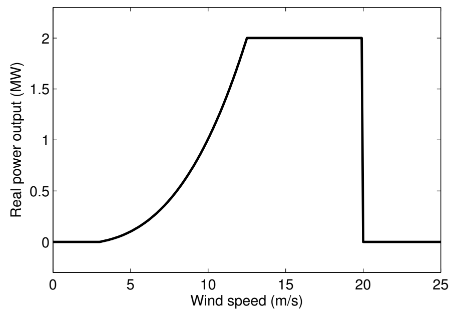


Figure 5.5: The relationship between wind speed and real power output

where v is the wind speed, $v_{\rm ci}$ is the cut-in wind speed, $v_{\rm ra}$ is the rated wind speed, $v_{\rm co}$ is the cut-out wind speed, and $P_{\rm ra}$ is the rated power of wind turbine. It is assumed that only one type of variable-speed wind turbines is used in this experiment. The rate power of wind turbine is set to 2 MW, the rated wind speed is set to 12.5 m/s, the cut-in and cut-out wind speed are set to 4 m/s and 20 m/s, respectively. The relationship between wind speed and real power output is illustrated in Fig. 5.5.

When wind power is penetrated into a power system, the power flow calculation will be affected by the characteristic of the wind turbines. Generally, multiple wind turbine generators are embedded into a power grid in the form of wind farms. Thus, the model of wind farm which aggregates multiple wind turbine generators (variable-speed wind turbine generators are concerned here) is required. The simplest variable-speed aggregate model to construct is the one that represents the wind farm as a single equivalent wind turbine generator. Since the aggregation procedure is applied to groups of similar wind turbines in areas of the power system, the equivalent wind turbine to the entire wind farm is a scale up of a single wind turbine, whose wind power becomes the sum of the wind power generated by all the wind turbines in the wind farm [154][155]. This aggregation procedures have been extensively applied to power system analysis, particularly to stability analysis of large power systems [156].

During normal steady-state operation, the wind farm can be considered as a PQ node or a PV node depending on the control strategy that the wind farm adopted [157]. In this paper, assume that the wind farm system is to realize the maximum power tracking through the mechanical control on the wind turbine blade pitch angle and the electrical control on the power converter [158]. Thus the active power output of the aggregated model of the wind farm is determined by $P_{\mathrm{wt}} \times N_{\mathrm{wt}}$, where N_{wt} is the number of wind turbines in the wind farm. Besides, in the case of one-machine equivalent, the wind farm is reactive-neutral with the entire transmission system in the connection point [156]. This implies that there is no reactive power exchange between the wind farm and the transmission system, with the control of reactive power in the wind farm. With the features of maximum active power tracking and reactive-neutral control, the wind farm embedded in the power system is considered as PQ node during power flow calculation.

5.2.2 Simulation studies

MOLA is fully compared with MOEA/D and NSGA-II, based on the modified IEEE 30-bus power system and new England test system, which are penetrated with wind power. The parameters setting of MOEA/D and NSGA-II follows the suggestions in [103] and [29] respectively. The parameters of MOLA do not affect the performance substantially when MOLA is applied to solve different optimisation problems, and they are chosen empirically and preset as follows: $\alpha = 0.8$; $\tau = 0.2$; $k = 4$; $I_{\mathrm{emax}} = 5$; $\lambda_1 = 0.5$; $\lambda_2 = 0.25$; $M = 50$; $D = 15$. To compare the algorithms fairly, the same N_{femax} is taken by the three algorithms when solving the same test case. The coefficients in (5.1.3) follow the suggestions given in [143].

The problem aims to gain a set of non-dominated solutions whose objective function vectors are evenly distributed on the Pareto front. The quality of the Pareto front is evaluated by two widely used measures, hypervolume indicator HV and Diversity metric $\triangle$, which are introduced in Section 3.4.1.

Modified IEEE 30-bus power system

The modified IEEE 30-bus power system consists of 41 branches and 6 fuel generators, as shown in Fig. 4.2. Nodes 7, 10, 16, 24 and 30 are incorporated with wind farms.

Table 5.8: The wind speed and number of wind turbines in the wind farms

Node	7	10	16	24	30
Wind speed (m/s)	9.3	15	7.6	8.7	6.5
Number of turbines	10	5	10	5	10

Static Environment: The wind speed and the number of wind turbines operated in the wind farms are given in Table 5.8. Control variables X include generator real power outputs, generator voltages, transformer tap settings and reactive power generations of the capacitor bank. N_{femax} is set to 30,000.

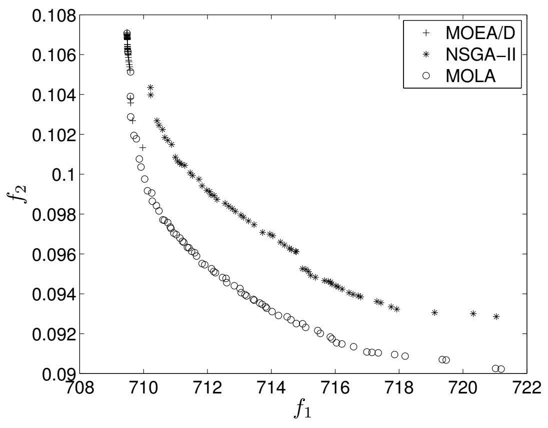


Figure 5.6: Pareto fronts obtained by MOEA/D, NSGA-II and MOLA on the IEEE 30-bus power system with wind power penetration

The Pareto fronts obtained by the three algorithms are presented in Fig. 5.6. It is obvious that MOLA outperforms NSGA-II in terms of finding accurate non-dominated solutions and wide range of the Pareto fronts. The non-dominated solutions obtained by MOEA/D and MOLA overlap in the objective space. However, the range of their Pareto fonts varies widely. To observe clearly, the Pareto fronts found by the two algorithms are plotted in separate subfigures, as shown in Fig 5.7. The non-dominated solutions found by MOEA/D flock to one end, situating at the range of [709.483, 709.97]×[0.1013, 0.1071]. Among the solutions found by MOEA/D, there is an isolated point, $i.e.$ (709.97, 0.1013). Besides, a big gap exists between solutions (709.66, 0.1027) and (709.97, 0.1013), which implies that MOEA/D has difficulty in finding the solutions located in the gap. As for MOLA, its non-dominated

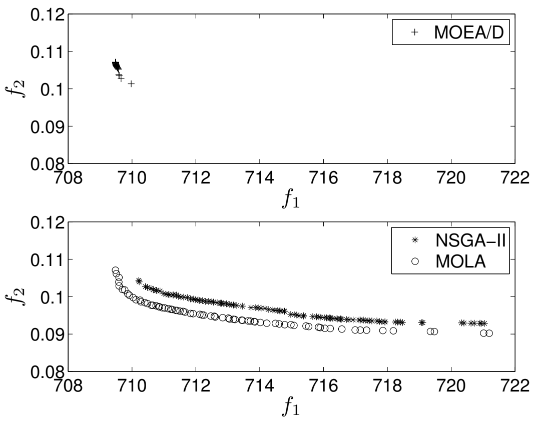


Figure 5.7: Pareto fronts (in separate subfigures) obtained by MOEA/D, NSGA-II and MOLA on the IEEE 30-bus power system with wind power penetration

solutions spread over the range of [709.484, 721.193]$\times$[0.0902, 0.1071], which is much wider than that of MOEA/D. Furthermore, the non-dominated solutions found by MOLA can dominate the isolated solution obtained by MOEA/D, $i.e.$ (709.97, 0.1013). MOLA can find more non-dominated solutions (224) than MOEA/D (101). The fact that MOLA finds wider Pareto front and more non-dominated solutions than MOEA/D suggests that MOLA can provide more options of the non-dominated solutions that satisfy the optimisation targets.

The reference solution and extreme solutions that are employed to calculate HV and Δ are listed in the first two rows of Table 5.9. The mean and standard deviation of HV and Δ obtained by these algorithms over 20 runs are also given in Table 5.9. It can be seen that MOLA outperforms MOEA/D and NSGA-II, as it finds larger HV and smaller Δ on average. In addition, the standard deviation of HV and Δ obtained by MOLA is consistently smaller than that obtained by MOEA/D and NSGA-II. This fact presents the reliability of MOLA in solving this problem.

Dynamic Environment: The demand for electricity and wind power fluctuate throughout the day. In order to simulate the time-varying operation condition, the whole optimisation process is divided into six stages, in which there are different power demand and wind speed. The fluctuating power demand is given in Table 5.10. Nodes 7, 10, 16, 24 and 30 are incorporated with wind farms, and the wind speed of these locations is given in Table 5.11. NSGA-II, MOEA/D and MOLA

Table 5.9: The setting of reference solution and extreme solutions; HV and Δ obtained by NSGA-II, MOEA/D and MOLA on the modified IEEE 30-bus power system (including mean and standard deviation)

		Reference solution		Extreme solution 1		Extreme solution 2	
Settings		[721.2, 0.108]		[709.48, 0.107]		[721.19, 0.090]	
Results		HV				Δ	
		Mean	Std			Mean	Std
	MOEA/D	0.0770	0.0041			1.0273	0.4212
	NSGA-II	0.1327	0.0075			0.1263	0.2138
	MOLA	0.1714	0.0027			0.1084	0.0139

Table 5.10: Load demand (MW) at bus 10

Stage	1	2	3	4	5	6
Load demand	2.5	3.5	5.5	4.7	5	3.8

are applied to solve the multi-objective problem in the modified IEEE 30-bus power system. 20,000 FEs are allowed in each stage.

Fig. 5.8 shows the hypervolume convergence characteristics obtained by the three algorithms, while the reference point is set to (700, 0.1). Notice that the higher the hypervolume values are, the better the Pareto front is. Between two stages, there is a sharp drop of the hypervolume values. This is because at the beginning of a new stage, the power system configuration is changed, and the objective function vectors

Table 5.11: Wind speed for the dynamic IEEE 30-bus power system

	Stage 1					Stage 2				
Nodes	7	10	16	24	30	7	10	16	24	30
Wind speed	5.6	10.3	9.7	8.4	17	7.3	8.6	10.9	11.3	12.3
	Stage 3					Stage 4				
Nodes	7	10	16	24	30	7	10	16	24	30
Wind speed	10.1	6.7	12.6	10.7	10.6	12.7	5.6	9.8	8.6	9.4
	Stage 5					Stage 6				
Nodes	7	10	16	24	30	7	10	16	24	30
Wind speed	9.4	7.1	8.4	10.3	7.9	10.5	9.0	6.9	9.3	10.3

need to be newly recorded. Besides, some of the non-dominated solutions obtained in the previous stage are not qualified as non-dominated solutions any more, and they are not suitable for the new operation condition. With the exploration strategies employed by the algorithms, the hypervolume values could be very small at the beginning of one stage until a larger hypervolume value is found. Among the three algorithms, MOEA/D and NSGA-II have a more serious drop of hypervolume values when a new stage occurs, which implies that the previously obtained solutions provide less inheritable information for the new stage, in terms of finding non-dominated solutions which are suitable for the new operation condition. It can be seen from Fig. 5.8 that MOLA has a faster converge rate in the first stage, compared with MOEA/D and NSGA-II. For the following stages, MOLA is able to track the changes of the power system configuration more accurately than the other two algorithms, and the hypervolume values obtained by MOLA are larger than those obtained by MOEA/D and NSGA-II. As for MOEA/D, its hypervolume values increase faster than NSGA-II in the first stage, and has similar performance as NSGA-II in the second stage. However, the performance of MOEA/D is much worse than that of NSGA-II in stages 3-6. As for NSGA-II, its convergence curve is almost flat within one stage (with the exception of stage 1). This implies that its search strategy applied to stages 2-6 does not take significant effect after the first change of the operation environment.

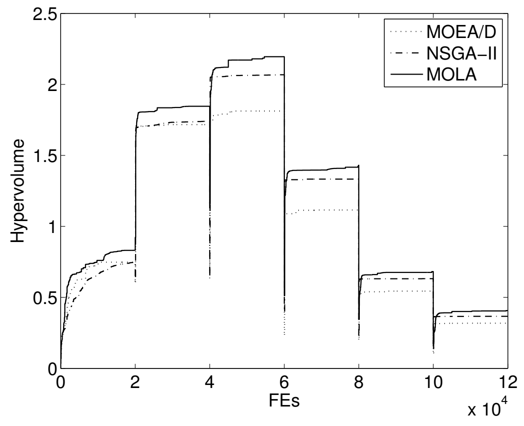


Figure 5.8: Convergence characteristics of MOEA/D, NSGA-II and MOLA on the IEEE 30-bus power system with wind power penetration

Table 5.12: The wind speed and number of wind turbines in the wind farms

Node	4	8	15	20	24	29
Wind speed (m/s)	7.9	11.2	10.1	9.3	8.6	16
Number of turbines	20	30	20	20	25	10

Modified New England Test System

A reduced model of the power system in new England is modified and penetrated with wind power, as given in Fig. 5.3. It consists of 46 branches and 10 fuel generators. Nodes 4, 8, 15, 20, 24 and 29 are incorporated with wind farms.

Static Environment: The wind speed and the number of wind turbines operated in the wind farms are given in Table 5.12. N_{femax} is set to 40,000 for MOEA/D, NSGA-II and MOLA.

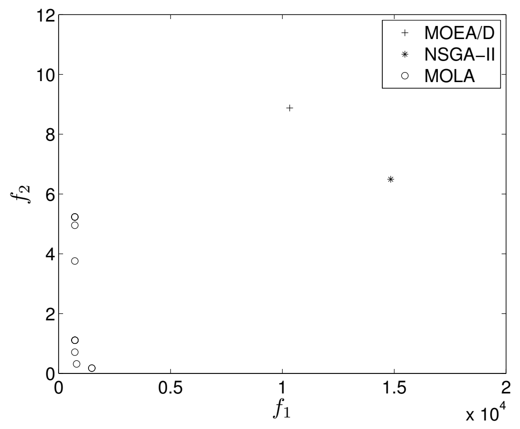


Figure 5.9: Pareto fronts obtained by MOEA/D, NSGA-II and MOLA on the modified new England wind power penetrated system

The Pareto fronts obtained by the three algorithms are given in Fig. 5.9. It can be seen that MOLA greatly presents its superiority over MOEA/D and NSGA-II with respect to the following two facts: the non-dominated solutions found by MOLA are much smaller than those found by MOEA/D and NSGA-II; the range of the Pareto front found by MOLA is much wider than that of MOEA/D and NSGA-II. It should be mentioned that MOEA/D and NSGA-II also find some solutions which locate out of the range given in the figure. However, considering the readability, they are not

Table 5.13: The setting of reference solution and extreme solutions; HV and Δ obtained by NSGA-II, MOEA/D and MOLA on the modified new England test system (including mean and standard deviation)

Settings	Reference solution	Extreme solution 1		Extreme solution 2	
	[14844, 8.877]	[722.7, 5.234]		[1482, 0.176]	
Results		HV		Δ	
		Mean	Std	Mean	Std
	MOEA/D	0.7978	0.598	9.2257×10^3	1.859×10^2
	NSGA-II	0.2082	0.2187	1.3742×10^4	7.684×10^3
	MOLA	1.227×10^5	1.7295×10^2	107.623	11.79

plotted in the figure, as they are far beyond the range given in this figure and all of them are dominated by the non-dominated solutions found by MOLA.

Table 5.13 lists the setting of reference solution and two extreme solutions, and the obtained values of HV and Δ (including mean and standard deviation over 20 runs). The results have confirmed the superiority of MOLA over MOEA/D and NSGA-II. It can be seen that MOLA finds much larger HV and smaller Δ than MOEA/D and NSGA-II on average. In addition, the standard deviation of Δ obtained by MOLA is smaller than that obtained by MOEA/D and NSGA-II.

Dynamic Environment: NSGA-II, MOEA/D and MOLA are applied to solve the multi-objective problem in the new England wind power penetrated system which is embedded with fluctuating power demand and wind power. The power demand of different stages is given in Table 5.10. The wind speed at nodes 4, 8, 15, 20, 24 and 29 varies according to Table 5.14. 20,000 FEs are allowed in one stage.

Fig. 5.10 provides the convergence curves of hypervolume values obtained by MOEA/D, NSGA-II and MOLA, while the reference point is set to (20,000, 15). It can be observed that the tracking ability of MOLA is constantly better than that of MOEA/D and NSGA-II as the function evaluation proceeds. The curves obtained by MOLA reveal that the variation of its hypervolume values is in line with the variation of the power system configuration. Unlike the case of IEEE 30-bus power system, the hypervolume values found by MOLA among different stages are similar in this test case, due to the fact that the percentage of wind power in the power system is much smaller and thus the variation of wind speed does not affect the

Table 5.14: Wind speed for dynamic new England wind power penetrated system

	Stage 1						Stage 2					
Nodes	4	8	15	20	24	29	4	8	15	20	24	29
Wind speed	8.4	10.9	10.5	9.7	9.4	11.5	7.4	9.5	11.3	8.6	8.9	10.5
	Stage 3						Stage 4					
Nodes	4	8	15	20	24	29	4	8	15	20	24	29
Wind speed	6.7	10.3	8.4	5.3	7.4	9.9	5.7	9.7	7.4	5.9	7.1	9.3
	Stage 5						Stage 6					
Nodes	4	8	15	20	24	29	4	8	15	20	24	29
Wind speed	4.5	7.8	6.9	6.3	7.8	8.9	5.6	7.3	7.9	5.9	8.9	10.4

hypervolume values significantly. For this power system, it is difficult for NSGA-II to respond to the changing frequency of the dynamic power system, as the maximum number of FEs constrained in each stage is not enough for NSGA-II to converge to better hypervolume values before the next stage comes. It can be seen that NSGA-II cannot converge to acceptable solutions until at the last stage. Besides, it responds to the changing system configuration with a serious delay in stages 1 and 2. As for MOEA/D, it is unable to converge to accurate solutions compared with MOLA and NSGA-II. It cannot obtain a satisfactory tracing performance.

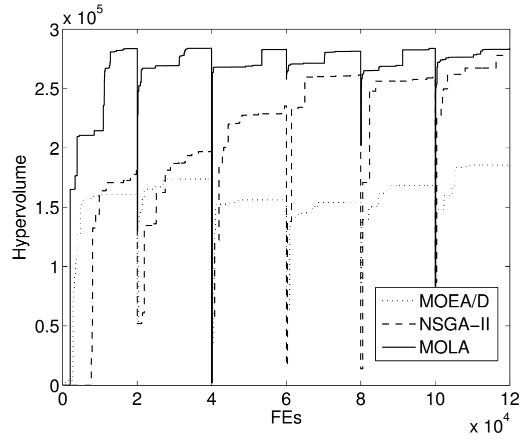


Figure 5.10: Convergence characteristics of MOEA/D, NSGA-II and MOLA on the new England wind power penetrated system penetrated with wind power

Remarking of Voltage Profile

Unlike MOEA/D and NSGA-II that change all the control variables simultaneously, MOLA mainly applies sequential search, which adjusts control variables one by one. Sequential search has the advantage of avoiding large perturbation of voltage, $i.e.$ maintaining the variation of voltage profile in a reasonable range. Keeping voltage profile stable is one of the most important constraints in terms of the implementation of an optimisation algorithm in power systems. In this aspect, MOLA has shown its superiority over MOEA/D and NSGA-II. To prove that, MOEA/D, NSGA-II and MOLA are applied to solve the multi-objective problem on the modified IEEE 30-bus power system with wind power penetration.

In the process of optimisation, the voltage profile is evaluated through an indicator of the perturbation of the voltage profile, denoted as PVP, which is defined as the difference of voltage profile between two continuous function evaluations, as formulated below:

$$PVP_i = \sum_{j\in N} |V_{i,j} - V_{i-1,j}| \tag{5.2.5}$$

where i denotes the index of FEs, and N denotes the set of numbers of the buses in the power system. PVP represents the perturbation of the voltage profile between two evaluations in the optimisation process. Smaller values of PVP imply stabler voltage profile. The values of PVP obtained over 31 continuous power flow calculations are given in Fig. 5.11, as well as its mean value. The mean value of PVP obtained by MOLA is 0.6133, while those of MOEA/D of NSGA-II are 0.7557 and 0.7473 respectively. It can be seen that the voltage profile obtained by MOLA is better than that found by MOEA/D and NSGA-II on average.

5.3 Evaluation in Deregulated Power Market

MOLA is applied to solve the problem in deregulated power market. The merits of MOLA have been demonstrated, in comparison with MOEA/D and NSGA-II. The study is undertaken based on the IEEE 30-bus power system. MOLA is applied to maximise social benefit and enhance the system stability simultaneously. The

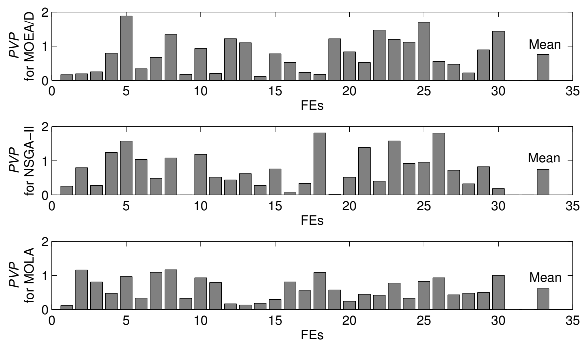


Figure 5.11: The values of PVP obtained by MOEA/D, NSGA-II and MOLA

simulation results have shown that MOLA is superior over MOEA/D and NSGA-II with respect to finding the range of the Pareto fronts, the efficiency of the search, and the accuracy of the obtained non-dominated solutions.

5.3.1 Problem formulation

The problem is concerned with the following two objectives. The first objective is to maximise the social profit, which is defined as customer benefit minus the cost of generators and the cost of pollutant emission [143]:

$$f_1 = \max \sum_{t=1}^{N} \left\{ \sum_{j=1}^{N_D} \tilde{B}_j(P^t_{\mathrm{D}_j}) - \sum_{i=1}^{N_G} (\tilde{C}_i(P^t_{\mathrm{G}_i}) + E_i) \right\} \tag{5.3.1}$$

where $\tilde{B}_j$ is the benefit (or bid) function of customer j [147]; $\tilde{C}_i$ is the cost (or bid) function of generator i; E_i is the cost of the pollutant emission generated by the ith generator, defined as (5.1.3).

The second objective is to enhance the voltage stability which is a major power system weakness resulting in severe detriments with economical, technical and social dimensions, as aforementioned in Section 4.2.1. Voltage stability assessment is carried out through a global indicator $L_{\max}$, as given in (4.2.11).

5.3.2 Simulation studies

MOLA is fully compared with MOEA/D and NSGA-II in solving the multi-objective optimisation problem introduced in Section 5.3.1. The parameter settings of MOEA/D and NSGA-II follow the suggestions in [103] and [29] respectively. The parameters of MOLA are pre-set using the same values as those given in Section 2.3.2, with the exception of $D = 20$, $M = 50$ and N_{femax}. To have a fair comparison, the same N_{femax} is taken by the algorithms when solving the same test case. The functions of bids and offers follow the suggestions in [147]. The comparison is carried out based on the IEEE 30-bus power system which consists of 48 branches and 6 generators, as shown in Fig. 4.2.

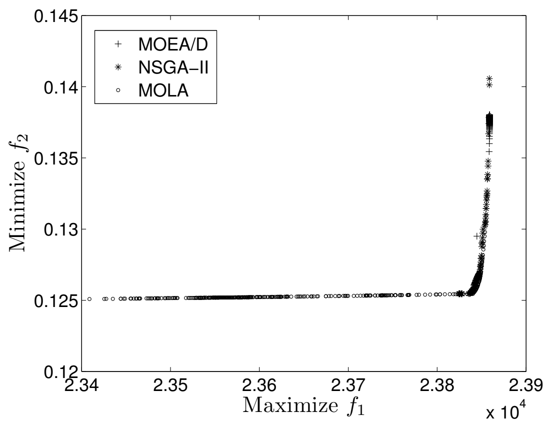


Figure 5.12: Pareto fronts obtained by MOEA/D, NSGA-II and MOLA on the IEEE 30-bus power system

The Pareto fronts obtained by MOEA/D, NSGA-II and MOLA are shown in Fig. 5.12, while Fig. 5.13 zooms in part of the Pareto front that locates within [2.378×10^4, 2.390×10^4]$\times$[0.125, 0.131]. It can be seen that MOLA performs better than MOEA/D and NSGA-II in terms of accuracy and smoothness of the Pareto front. In addition, the range of their Pareto fonts varies widely. To see clearly, the results of the three algorithms are plotted in subfigures separately, as shown in Fig 5.14. The non-dominated solutions found by MOEA/D flock to one end, locating at the range of [2.384×10^4, 2.386×10^4]$\times$[0.1295, 0.138]. Among the non-dominated solutions found by MOEA/D, there is an isolated objective function vector, $i.e.$ (2.384×10^4, 0.1295). Besides, there is a big gap between solutions (2.384×10^4, 0.1295) and (2.386×10^4, 0.1354), which means that MOEA/D can not find any solution between the two points. The Pareto front found by NSGA-II spreads over [2.382×10^4, 2.386×10^4]$\times$[0.1255, 0.1378], which is wider than that found by MOEA/D. However, the non-dominated solutions found by MOEA/D are more accurate than those found by NSGA-II, with the exception of the isolated point, (2.384×10^4, 0.1295). As for MOLA, its non-dominated solutions spread over the range of [2.341×10^4, 2.386×10^4]$\times$[0.1251, 0.138], which is much wider than that of MOEA/D and NSGA-II. Additionally, it can be seen that MOLA can find much more non-dominated solutions than MOEA/D and NSGA-II. The fact that MOLA finds wider Pareto front and more non-dominated solutions suggests that MOLA can provide a variety of choices among the non-dominated solutions that satisfy the optimisation targets.

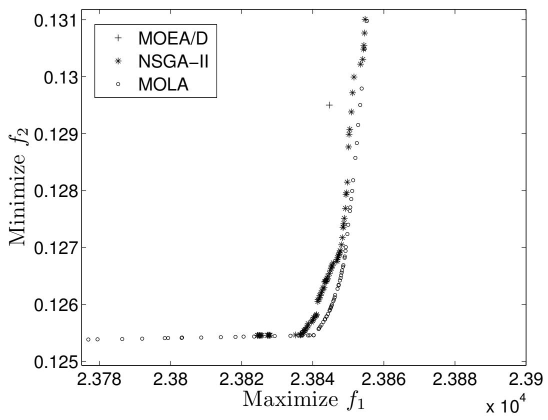


Figure 5.13: Details of the Pareto fronts obtained by MOEA/D, NSGA-II and MOLA on the IEEE 30-bus power system

The computation time consumed by the three algorithms is also investigated on the power system, as given in Table 5.15, which lists the computation time used by the three algorithms. It can be seen that MOLA reduces the computation time, in comparison with MOEA/D and NSGA-II.

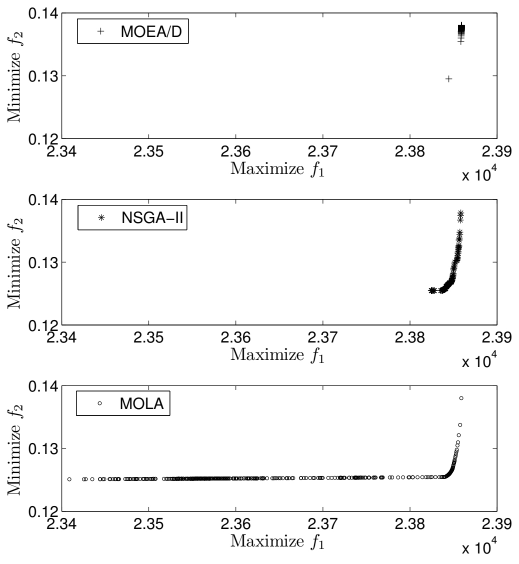


Figure 5.14: Pareto fronts obtained by MOEA/D, NSGA-II and MOLA (in subfigures) on the IEEE 30-bus power system

Table 5.15: The computation time (s) consumed by MOEA/D, NSGA-II and MOLA on the IEEE 30-bus power system

Algorithms	MOEA/D	NSGA-II	MOLA
Time (s)	148.462	201.243	123.514

5.4 Conclusions

In this chapter, MOLA has been fully compared with MOEA/D and NSGA-II in solving the multi-objective optimisation problem, which aims to reduce the operational cost and enhance voltage stability, in the wind power penetrated IEEE 30-bus power system and new England test power system respectively. The simulation results have demonstrated that MOLA performs better than MOEA/D and NSGA-II, as MOLA can find wider and evenly distributed Pareto fronts, and obtain more accurate Pareto optimal solutions efficiently. Additionally, MOLA consistently finds larger hypervolume and smaller diversity metric than MOEA/D and NSGA-II under different circumstances. MOLA has presented its superiority by finding wider Pareto fronts than MOEA/D and obtaining more accurate solutions than NSGA-II, while using much less function evaluations.

Besides, MOLA has also been applied to solve the multi-objective optimisation problem in deregulated market, which aims to maximise the social benefit and enhance voltage stability in the IEEE 30-bus power system. The simulation results have demonstrated that MOLA greatly outperforms MOEA/D and NSGA-II, as MOLA can find wider and evenly distributed Pareto fronts, and obtain more accurate Pareto optimal solutions efficiently.

Chapter 6

Conclusions and Future Work

This chapter concludes this thesis, summarises the major outcomes of the research work presented in this thesis, and indicates possible future directions for further investigation in terms of learning automata methods and their applications in power systems.

6.1 Conclusions

In the literature of opitmisation, research of EAs has attracted great attention due to their ability of solving multi-modal problems. However, based on the population-based search approach, these algorithms have a high level of randomness and unpredictable computational complexity. This has hampered the potential of these approaches' application in large-scale optimisation problems, such as optimal power flow problems in power systems. The research here is to develop an alternative approach to function optimisation, based on learning automata, instead of EAs. In this thesis, two novel optimisation algorithms, FOLA and MOLA, have been developed. Following the methodological studies, the developed algorithms have been applied to resolve the optimal power flow problems in power systems. The research work presented in this thesis can be concluded as follows.

The first part of the thesis (including Chapters 2 and 3) is devoted to the development of the two novel learning automata-based algorithms, FOLA and MOLA.

In Chapter 2, FOLA is presented based on the concept of learning automata. It adopts the structure of multiple automata, and each automaton undertakes dimensional search on a selected dimension of the solution domain. FOLA capitalises on the merits of dividing the dimensional search domain into cells, and the memories of the performance evaluation of the dimensional states. With these approaches, FOLA is able to undertake search in continuous states and achieve accurate solutions efficiently. There are two key parameters, w_{c} and N_{femax}, to tune in FOLA when it is applied to resolve a specific application problem. The two parameters can be determined with the basic knowledge of the range of variables involved and the solution accuracy required in the application problem. FOLA has been compared with a number of EAs based on widely used 30-dimensional benchmark functions which include uni-modal and multi-modal problems. The experimental results have shown that FOLA is able to achieve more accurate results than other EAs in finding a global minimum solution, and is more reliable, for its standard deviation of the results over different independent runs is much smaller than that of other algorithms. The capability of FOLA is also investigated based on high dimensional optimisation problems, whose dimensionality is up to 300. In comparison with other EAs, FOLA presents its great superiority, as it finds much more accurate solutions, and significantly improves the performance in terms of efficiency, convergence rate and computation load.

As an extension of FOLA, MOLA is presented to solve multi-objective optimisation problems in Chapter 3. MOLA employs the continuous search and the process of learning from the best solution and neighboring solutions, through which MOLA is able to find evenly distributed Pareto front for complex optimisation problems. MOLA has been fully compared with two popular weighted-sum algorithms, MOGA and MOPSO, on four multi-objective benchmark functions that comprise low and high-dimensional models, convex and non-convex models, and continuous and discontinuous models respectively. MOLA finds more accurate non-dominated solutions than the other two algorithms. The solutions found by MOGA and MOPSO flock to the two ends of the Pareto fronts, when solving the problems which have non-convex Pareto front. As the dimensionality increases, MOLA exhibits great superiority over MOGA and MOPSO with respect to the quality of Pareto fronts and computation time. With less computation time, the solutions found by MOLA can dominate most of the solutions found by MOGA and MOPSO. MOLA has been also compared with two Pareto front-based multi-objective algorithms, MOEA/D and NSGA-II, on the basis of thirteen widely used multi-objective benchmark functions, which comprise complex Pareto set shapes. The simulation results have shown that MOLA presents its superiority over MOEA/D and NSGA-II, as it can find more accurate and evenly distributed non-dominated solutions than MOEA/D and NSGA-II, and its Pareto fronts are wider than those obtained by MOEA/D and NSGA-II.

The second part of the thesis (including Chapters 4 and 5) is devoted to the applications of FOLA and MOLA in power systems. In the first application, FOLA is applied to reduce the fuel cost and enhance the voltage stability of the power system. Simulation studies are carried out on the standard IEEE 30-bus and 57-bus power systems respectively. The simulation results have demonstrated that FOLA is able to offer more accurate solutions with shorter computation times. In the second application, FOLA is applied to solve the optimal power flow problem which aims to achieve economic power system dispatch and voltage stability enhancement in dynamic wind power integrated systems. FOLA is compared with both classical and recently-proposed EAs respectively, on the basis of the modified IEEE 30-bus, 57-bus power and 118-bus systems, which are integrated with time-varying wind power. The experimental results have demonstrated that FOLA tracks the changing system configuration more rapidly and accurately than other algorithms. Advantages of FOLA have been demonstrated by the fact that FOLA reduces the fuel cost greatly and enhances the voltage stability of the power system.

Chapter 5 demonstrated three applications of MOLA for power system optimisation: solving economic emission dispatch and voltage stability enhancement in wind power integrated power systems, and optimising social benefit and voltage stability in deregulated electricity market. In the first two applications, a brief introduction to the problem formulation, followed by the experiment undertaken on the modified IEEE 30-bus power system and the modified new England test system, which are integrated with fixed-speed and variable speed wind generators respectively. The dynamic environment is also investigated in this case. The simulation results have presented the superiority of MOLA over the other algorithms. In the third application, with the emergence of deregulated electricity markets, the challenge and modification of optimal power flow are introduced first. MOLA is applied to maximise social benefit and enhance voltage stability in deregulated electricity market. MOLA is compared with MOEA/D and NSGA-II, in solving the challenging optimisation problems, based on IEEE 30-bus power system. The simulation results have presented the superiority of MOLA, as MOLA can find wide and evenly distributed Pareto fronts, and obtain accurate non-dominated solutions.

Conclusively, from the successful developments of the two novel algorithms, this thesis demonstrates the prospects of the studies of LA-based algorithms. FOLA and MOLA provide a new vision for optimisation problems. This thesis also demonstrates the outstanding performance of these two algorithms while they are applied to solve power system optimisation problems.

6.2 Suggestions for Future Work

The thesis aims at the development of LA-based optimisation algorithms and to fully explore their potential for applications in power systems. In this section, suggestions for future work that aim at improving the algorithms and expanding their application are provided.

- The reinforcement signals (or cell values) could be extended into multiple dimensions, thus they can represent the information collected from different perspectives. For instance, if the search domain is small enough, the problem could be considered as uni-modal. It is well-known that traditional gradient-based optimisation algorithms can solve uni-modal problems accurately, therefore, the approximated gradient could be regarded as one dimension of the reinforcement signals (or cell values). In addition, the correlation among variables can also be taken into account as one dimension. With multiple dimensions of the collected information available, learning automata can automatically combine the beneficial information to guide the search around more prosperous areas, based on the property of the searching area. The future work will concentrate on the investigation of the structure of multiple dimensional reinforcement signals (or cell values), and then generating an automatic strategy which can analyse the property of the searching area, and choose the corresponding useful information as search guidance.

- Preliminary study on hierarchical systems of learning automata has shown the merit of using the structure of multilevel hierarchy, which is attractive to the problems which require a large number of actions [69]. The structure has the potential ability to tackle multi-objective problems. The future work will investigate the hierarchical system of learning automata and construct a framework of three level hierarchy, in which the learning automata located in the lowest level interact with the environment, the learning automata located in the middle level are used to resolve the multiple objectives separately, and the learning automata located in the highest level are used to make trade-off among the multiple objectives.

- Several inspiring learning methods (or learning theories) have been developed in the literature, for instance, reinforcement learning [82] and statistical learning. Although these methods are limited to deal with certain types of problem, their learning behaviors will be studied, which could lead to the discovery of potential behaviors that benefits optimisation. These learning behavior could afterwards be modified and adopted in the proposed methods, in order to further improve the efficiency and accuracy of the algorithms.

- This study will expand the application of FOLA and MOLA in complex optimisation problems in power systems. The gap between the simulation and on-line implementation is still wide. Applying the proposed algorithms in power systems on-line still requires a large amount of work. In further work, the optimisation problems of power systems which required on-line implementation will be investigated and solved by the proposed algorithms.

Appendix A

Benchmark Functions

A.1 Unimodal Benchmark Functions

Sphere Model

$$F_1(x) = \sum_{i=1}^{N} x_i^2 \qquad -100 \le x_i \le 100$$
$$\min(F_1) = F_1(0, ..., 0) = 0$$

Schwefel's Problem 2.22

$$F_2(x) = \sum_{i=1}^{N} |x_i| + \prod_{i=1}^{N} |x_i| \qquad -10 \le x_i \le 10$$
$$\min(F_2) = F_2(0, ..., 0) = 0$$

Schwefel's Problem 1.2

$$F_3(x) = \sum_{i=1}^{N} \left(\sum_{j=1}^{i} x_j \right) \qquad -100 \le x_i \le 100$$
$$\min(F_3) = F_3(0, ..., 0) = 0$$

Schwefel's Problem 2.21

$$F_4(x) = \max_i \{|x_i|, 1 < i < N\} \qquad -100 \le x_i \le 100$$
$$\min(F_4) = F_4(0, ..., 0) = 0$$

Generalised Rosenbrock's Function

$$F_5(x) = \sum_{i=1}^{N-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)]^2; \qquad -5 \le x_i \le 5$$
$$\min(F_5) = F_5(1, ..., 1) = 0$$

Step Function

$$F_6(x) = \sum_{i=1}^{N}(\lfloor x_i + 0.5 \rfloor)^2 \qquad -100 \le x_i \le 100$$
$$\min(F_6) = F_6(0, ..., 0) = 0$$

Quartic Function with Noise

$$F_7(x) = \sum_{i=1}^{N} i x_i^4 + \texttt{random}[0, 1) \qquad -1.28 \le x_i \le 1.28$$
$$\min(F_7) = F_7(0, ..., 0) = 0$$

A.2 Multimodal Benchmark Functions

Generalised Schwefel's Problem 2.26

$$F_8(x) = \sum_{i=1}^{N} -x_i \sin(\sqrt{|x|}) \qquad -500 \le x_i \le 500$$
$$\min(F_8) = F_8(420.9687, ..., 420.9687) = -12,569.5$$

Generalised Rastrigin's Function

$$F_9(x) = \sum_{i=1}^{N}[x_i^2 - 10\cos(2\pi x_i) + 10]^2 \quad -5.12 \le x_i \le 5.12$$
$$\min(F_9) = F_9(0, ..., 0) = 0$$

Ackley's Function

$$\begin{aligned} F_{10}(x) &= -20\exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}\right) - \exp\left(\frac{1}{N}\sum_{i=1}^{N}\cos(2\pi x_i)\right) + 20 + e \\ & -32 \le x_i \le 32 \\ & \min(F_{10}) = F_{10}(0, ..., 0) = 0 \end{aligned}$$

Generalised Girewank Function

$$F_{11}(x) = \frac{1}{4000}\sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

$$-600 \le x_i \le 600$$

$$\min(F_{11}) = F_{11}(0, ..., 0) = 0$$

Generalised Penalised Function 1

$$\begin{aligned} F_{12}(x) &= \frac{\pi}{N}\{10\sin^2(\pi y_1) + \sum_{i=1}^{N-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_{N-1})^2\} \\ &\quad + \sum_{i=1}^{N} u(x_i, 5, 100, 4), \\ &\quad -50 \le x_i \le 50 \\ &\quad \min(F_{12}) = F_{12}(1, ..., 1) = 0 \end{aligned}$$

where $y_i = 1 + \frac{1}{4}(x_i + 1),$

$$u(x_i, a, k, m) = \begin{cases} k(x_i - 1)^m, & x_i > a, \\ 0, & -a < x_i < a, \\ k(-x_i - 1)^m, & x_i < -a. \end{cases}$$

Generalised Penalised Function 2

$$\begin{aligned} F_{13}(x) &= 0.1\{\sin^2(3\pi x_1) + \sum_{i=1}^{N-1}(x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})] \\ &\quad +(x_N - 1)^2[1 + \sin^2(2\pi x_N)]\} + \sum_{i=1}^{N} u(x_i, 5, 100, 4), \\ &\quad -50 \le x_i \le 50 \\ &\quad \min(F_{13}) = F_{13}(1, ..., 1) = 0 \end{aligned}$$

where $y_i = 1 + \frac{1}{4}(x_i + 1),$

$$u(x_i, a, k, m) = \begin{cases} k(x_i - 1)^m, & x_i > a, \\ 0, & -a < x_i < a, \\ k(-x_i - 1)^m, & x_i < -a. \end{cases}$$

A.3 Multimodal Benchmark Functions with Rotation and Shift

Shifted Rosenbrock's Function

$$
\begin{aligned}
F_{\mathrm{rs1}}(x) &= \sum_{i=1}^{N-1} \left[100(z_{i+1} - z_i^2)^2 + (z_i - 1)^2\right] + f_{\mathrm{bias}} \\
& Z = X - O + 1 \\
& X = [x_1, \ldots, x_N] \\
& N = 30; \;\; -100 \leq x_i \leq 100 \\
& \min(F_{\mathrm{rs1}}) = F_{\mathrm{rs1}}(o_1, \ldots, o_N) = 390
\end{aligned}
$$

Shifted Rotated Griewank's Function without Bounds

$$
\begin{aligned}
F_{\mathrm{rs2}}(x) &= \frac{1}{4000}\sum_{i=1}^{N} z_i^2 - \prod_{i=1}^{N} \cos\left(\frac{z_i}{\sqrt{i}}\right) + 1 + f_{\mathrm{bias}} \\
& Z = (X - O) \times M \\
& M = M'(1 + 0.3|N(0,1)|); \;\; M' \text{ is linear transformation matrix} \\
& N = 30; \;\; 0 \leq x_i \leq 600 \\
& \min(F_{\mathrm{rs2}}) = F_{\mathrm{rs2}}(o_1, \ldots, o_N) = -180
\end{aligned}
$$

Shifted Rotated Ackley's Function with Global Optimum on Bounds

$$
\begin{aligned}
F_{\mathrm{rs3}}(x) &= -20 \exp\big(-0.2\sqrt{1/N \sum_{i=1}^{N} z_i^2}\big) - \exp\big(1/N \sum_{i=1}^{N} \cos(2\pi z_i)\big) \\
& +20 + \mathrm{e} + f_{\mathrm{bias}} \\
& Z = (X - O) \times M \\
& M \text{ is linear transformation matrix} \\
& N = 30; \;\; -32 \leq x_i \leq 32 \\
& \min(F_{\mathrm{rs3}}) = F_{\mathrm{rs3}}(o_1, \ldots, o_N) = -140
\end{aligned}
$$

Shifted Rastrigin's Function

$$\begin{aligned} F_{\text{rs4}}(x) &= f_4 = \sum_{i=1}^{N} \big(z_i^2 - 10\cos(2\pi z_i) + 10\big) + f_{\text{bias}} \\ & Z = (X - O) \\ & N = 30;\ \ -5 \leq x_i \leq 5 \\ & \min(F_{\text{rs4}}) = F_{\text{rs4}}(o_1, \ldots, o_N) = -330 \end{aligned}$$

Shifted Rotated Rastrigin's Function

$$\begin{aligned} F_{\text{rs5}}(x) &= \sum_{i=1}^{N} \big(z_i^2 - 10\cos(2\pi z_i) + 10\big) + f_{\text{bias}} \\ & Z = (X - O) \times M \\ & M \text{ is linear transformation matrix} \\ & N = 30;\ \ -5 \leq x_i \leq 5 \\ & \min(F_{\text{rs5}}) = F_{\text{rs5}}(o_1, \ldots, o_N) = -330 \end{aligned}$$

Shifted Rotated Weierstrass Function

$$\begin{aligned} F_{\text{rs6}}(x) &= \sum_{i=1}^{N} \Big(\sum_{k=0}^{k\max} [a^k \cos(2\pi b_k (z_i + 0.5))] \Big) \qquad \text{(A.3.1)} \\ & -N \sum_{k=0}^{k\max} [a^k \cos(2\pi b_k \cdot 0.5)] + f_{\text{bias}} \\ & a = 0.5;\ b = 3;\ k\max = 20; \\ & Z = (X - O) \times M \\ & M \text{ is linear transformation matrix} \\ & N = 30;\ \ -0.5 \leq x_i \leq 0.5 \\ & \min(F_{\text{rs6}}) = F_{\text{rs6}}(o_1, \ldots, o_N) = 90 \end{aligned}$$

Schwefel's Problem 2.13

$$\begin{aligned} F_{\text{rs7}}(x) &= \sum_{i=1}^{N}(A_i - B_i(x))^2 + f_{\text{bias}} \\ & a = 0.5;\ b = 3;\ k\max = 20; \\ & A_i = \sum_{j=1}^{N}(a_{ij}\sin\alpha + b_{ij}\cos\alpha) \\ & B_i(x) = \sum_{j=1}^{N}(a_{ij}\sin x_j + b_{ij}\cos x_j) \\ & N = 30;\ \ -\pi \le x_i \le \pi \\ & \min(F_{\text{rs7}}) = F_{\text{rs7}}(o_1, \ldots, o_N) = -460 \end{aligned}$$

Expanded Extended Griewank's plus Rosenbrock's Function

$$\begin{aligned} F_{\text{rs8}}(x) &= \dot{F}(\ddot{F}(z_1, z_2)) + \dot{F}(\ddot{F}(z_2, z_3)) + \ldots + \dot{F}(\ddot{F}(z_{N-1}, z_N)) \\ & +\dot{F}(\ddot{F}(z_N, z_1)) + f_{\text{bias}} \\ & \dot{F} = \frac{1}{4000}\sum_{i=1}^{N} z_i^2 - \prod_{i=1}^{N}\cos\left(\frac{z_i}{\sqrt{i}}\right) + 1 \\ & \ddot{F} = \sum_{i=1}^{N-1}\left[100(z_{i+1} - z_i^2)^2 + (z_i - 1)^2\right] \\ & Z = X - O + 1 \\ & N = 30;\ \ -\pi \le x_i \le \pi \\ & \min(F_{\text{rs8}}) = F_{\text{rs8}}(o_1, \ldots, o_N) = -130 \end{aligned}$$

Expanded Rotated Extended Scaffe's F6

$$\begin{aligned} F_{\text{rs9}}(x) &= \dot{F}(z_1, z_2) + \dot{F}(z_2, z_3) + \ldots + \dot{F}(z_{N-1}, z_N) + \dot{F}(z_N, z_1) + f_{\text{bias}} \\ & \dot{F}(x, y) = 0.5 + \frac{(sin^2(\sqrt{x^2 + y^2}) - 0.5)}{(1 + 0.001(x^2 + y^2))^2} \\ & Z = (X - O) \times M \\ & M \text{is linear transformation matrix} \\ & N = 30;\ \ -100 \le x_i \le 100 \\ & \min(F_{\text{rs9}}) = F_{\text{rs9}}(o_1, \ldots, o_N) = -300 \end{aligned}$$

A.4 Multi-objective Benchmark Functions for Weighted-sum Based Algorithms

Multi-objective benchmark function I

$$\begin{aligned} f_1(X) &= \left(\sum_{i=1}^{N}(x_i^2)\right)^{1/8} \\ f_2(X) &= \left(\sum_{i=1}^{N}(x_i-0.5)^2\right)^{1/4} \\ & -5 \leq x_i \leq 5 \end{aligned}$$

Multi-objective benchmark function II

$$\begin{aligned} f_1(X) &= 1-\exp\left(-\sum_{i=1}^{N}(x_i-\frac{1}{\sqrt{N}})^2\right) \\ f_2(X) &= 1-\exp\left(-\sum_{i=1}^{N}(x_i+\frac{1}{\sqrt{N}})^2\right) \\ & -4 \leq x_i \leq 4 \end{aligned}$$

Multi-objective benchmark function III

$$\begin{aligned} f_1(X) &= x_1 \\ f_2(X) &= (1+10x_2)\left(1-\left(\frac{x_1}{1+10x_2}\right)^2-\frac{x_1\sin(8\pi x_1)}{1+10x_2}\right) \\ & 0 \leq x_1, x_2 \leq 1 \end{aligned}$$

Multi-objective benchmark function IV

$$\begin{aligned} f_1(X) &= 4.9\times 10^{-5}(x_2^2-x_1^2)(x_4-1) \\ f_2(X) &= \frac{9.82\times 10^{6}(x_2^2-x_1^2)}{x_3x_4(x_2^3-x_1^3)} \\ & (x_2-x_1)-20 \geq 0 \\ & 30-2.5(x_4+1) \geq 0 \\ & 0.4-\frac{x_3}{3.14(x_2^2-x_1^2)} \geq 0 \\ & \frac{2.22\times 10^{-3}x_3(x_2^3-x_1^3)}{(x_2^2-x_1^2)^2} \leq 1 \\ & \frac{2.66\times 10^{-2}x_3x_4(x_2^3-x_1^3)}{x_2^2-x_1^2} \geq 900 \end{aligned}$$

A.5 Multi-objective Benchmark Functions for Pareto front-based Algorithms

Group 1

Fun1-1

$$\begin{aligned} f_1(X) &= \left(\sum_{i=1}^{30}(x_i^2)\right)^{1/8} \\ f_2(X) &= \left(\sum_{i=1}^{30}(x_i-0.5)^2\right)^{1/4} \\ & -5 \le x_i \le 10 \end{aligned}$$

Fun1-2

$$\begin{aligned} f_1(X) &= \left(\sum_{i=1}^{50}(x_i^2)\right)^{1/8} \\ f_2(X) &= \left(\sum_{i=1}^{50}(x_i-0.5)^2\right)^{1/4} \\ & -5 \le x_i \le 10 \end{aligned}$$

Fun1-3

$$\begin{aligned} f_1(X) &= \left(\sum_{i=1}^{70}(x_i^2)\right)^{1/8} \\ f_2(X) &= \left(\sum_{i=1}^{70}(x_i-0.5)^2\right)^{1/4} \\ & -5 \le x_i \le 10 \end{aligned}$$

Fun2-1

$$\begin{aligned} f_1(X) &= 1-\exp\left(-\sum_{i=1}^{N}(x_i-\frac{1}{\sqrt{N}})^2\right) \\ f_2(X) &= 1-\exp\left(-\sum_{i=1}^{N}(x_i+\frac{1}{\sqrt{N}})^2\right) \\ & -4 \le x_i \le 4 \end{aligned}$$

Fun2-2

$$\begin{aligned} f_1(X) &= 1 - \exp\left(-\sum_{i=1}^{N}(x_i - \frac{1}{\sqrt{N}})^2\right) \\ f_2(X) &= 2.5 - 2.3 \times \exp\left(-\sum_{i=1}^{N}(x_i + \frac{1}{\sqrt{N}})^2\right) \\ & -4 \le x_i \le 4 \end{aligned}$$

Fun2-3

$$\begin{aligned} f_1(X) &= 1 - \exp\left(-\sum_{i=1}^{N}(x_i - \frac{1}{\sqrt{N}})^2\right) \\ f_2(X) &= 1 - \exp\left(-\sum_{i=1}^{N}(x_i + \frac{1}{\sqrt{N}})^6\right) \\ & -4 \le x_i \le 4 \end{aligned}$$

Group 2

Fun3

$$\begin{aligned} f_1(X) &= x_1 + \frac{2}{|J_1|}\sum_{j \in J_1}[x_j - \sin(6\pi x_1 + \frac{j\pi}{n})]^2 \\ f_2(X) &= 1 - \sqrt{x_1} + \frac{2}{|J_2|}\sum_{j \in J_2}[x_j - \sin(6\pi x_1 + \frac{j\pi}{n})]^2 \\ & J_1 = \{j | j \text{ is odd and } 2 \le j \le n\} \\ & J_2 = \{j | j \text{ is even and } 2 \le j \le n\} \\ & n = 30; \quad [0, 1] \times [-1, 1]^{n-1} \end{aligned}$$

Fun4

$$\begin{aligned}
f_1(X) &= x_1 + \frac{2}{|J_1|}\sum_{j\in J_1} y_j^2 \\
f_2(X) &= 1-\sqrt{x_1} + \frac{2}{|J_2|}\sum_{j\in J_2} y_j^2 \\
& J_1 = \{j|j \text{ is odd and } 2 \le j \le n\} \\
& J_2 = \{j|j \text{ is even and } 2 \le j \le n\} \\
y_j &= \begin{cases} x_j - [0.3x_1^2\cos(24\pi x_1 + \frac{4j\pi}{n}) + 0.6x_1]\cos(6\pi x_1 + \frac{j\pi}{n}) & j \in J_1 \\ x_j - [0.3x_1^2\cos(24\pi x_1 + \frac{4j\pi}{n}) + 0.6x_1]\sin(6\pi x_1 + \frac{j\pi}{n}) & j \in J_2 \end{cases} \\
& n = 30; \quad [0,1]\times[-1,1]^{n-1}
\end{aligned}$$

Fun5

$$\begin{aligned}
f_1(X) &= x_1 + \frac{2}{|J_1|}(4\sum_{j\in J_1} y_j^2 - 2\prod_{j\in J_1}\cos(\frac{20y_j\pi}{\sqrt{j}}) + 2) \\
f_2(X) &= 1-\sqrt{x_1} + \frac{2}{|J_2|}(4\sum_{j\in J_2} y_j^2 - 2\prod_{j\in J_2}\cos(\frac{20y_j\pi}{\sqrt{j}}) + 2) \\
& J_1 = \{j|j \text{ is odd and } 2 \le j \le n\} \\
& J_2 = \{j|j \text{ is even and } 2 \le j \le n\} \\
& y_j = x_j - x_1^{0.5(1.0+\frac{3(j-2)}{n-2})}, j = 2,\ldots,n \\
& n = 30; \quad [0,1]^n
\end{aligned}$$

Fun6

$$\begin{aligned}
f_1(X) &= x_1 + \frac{2}{|J_1|}(4\sum_{j\in J_1} h(y_j)) \\
f_2(X) &= 1-x_1^2 + \frac{2}{|J_2|}(4\sum_{j\in J_2} h(y_j)) \\
& J_1 = \{j|j \text{ is odd and } 2 \le j \le n\} \\
& J_2 = \{j|j \text{ is even and } 2 \le j \le n\} \\
& y_j = x_j - \sin(6\pi x_1 + \frac{j\pi}{n}), j = 2,\ldots,n \\
& n = 30; \quad [0,1]\times[-2,2]^{n-1}
\end{aligned}$$

Fun7

$$\begin{aligned} f_1(X) &= x_1 + (\frac{1}{2N} + \varepsilon)|\sin(2N\pi x_1)| + \frac{2}{|J_1|}\sum_{j\in J_1} h(y_j) \\ f_2(X) &= 1 - x_1 + (\frac{1}{2N} + \varepsilon)|\sin(2N\pi x_1)| + \frac{2}{|J_2|}\sum_{j\in J_2} h(y_j) \\ & J_1 = \{j|j \text{ is odd and } 2 \leq j \leq n\} \\ & J_2 = \{j|j \text{ is even and } 2 \leq j \leq n\} \\ & y_j = x_j - \sin(6\pi x_1 + \frac{j\pi}{n}), j = 2,\ldots,n \\ & h(t) = 2t^2 - \cos(4\pi t) + 1 \\ & n = 30; \quad [0,1] \times [-1,1]^{n-1} \end{aligned}$$

Fun8

$$\begin{aligned} f_1(X) &= x_1 + \max\{0, 2(\frac{1}{2N} + \varepsilon)\sin(2N\pi x_1)\} \\ & \quad + \frac{2}{|J_1|}(4\sum_{j\in J_1} y_j^2 - 2\prod_{j\in J_1}\cos(\frac{20y_j\pi}{\sqrt{j}}) + 2) \\ f_2(X) &= 1 - x_1 + \max\{0, 2(\frac{1}{2N} + \varepsilon)\sin(2N\pi x_1)\} \\ & \quad + \frac{2}{|J_2|}(4\sum_{j\in J_2} y_j^2 - 2\prod_{j\in J_2}\cos(\frac{20y_j\pi}{\sqrt{j}}) + 2) \\ & J_1 = \{j|j \text{ is odd and } 2 \leq j \leq n\} \\ & J_2 = \{j|j \text{ is even and } 2 \leq j \leq n\} \\ & y_j = x_j - \sin(6\pi x_1 + \frac{j\pi}{n}), j = 2,\ldots,n \\ & n = 30; \quad [0,1] \times [-1,1]^{n-1} \end{aligned}$$

Fun9

$$\begin{aligned} f_1(X) &= \sqrt[5]{x_1} + (\frac{2}{|J_1|}\sum_{j\in J_1} y_j^2 \\ f_2(X) &= 1 - \sqrt[5]{x_1} + (\frac{2}{|J_2|}\sum_{j\in J_2} y_j^2 \\ & J_1 = \{j|j \text{ is odd and } 2 \leq j \leq n\} \\ & J_2 = \{j|j \text{ is even and } 2 \leq j \leq n\} \\ & y_j = x_j - \sin(6\pi x_1 + \frac{j\pi}{n}), j = 2,\ldots,n \\ & n = 30; \quad [0,1] \times [-1,1]^{n-1} \end{aligned}$$

Group 3

Fun10

$$\begin{aligned}
f_1(X) &= \cos(0.5x_1\pi)\cos(0.5x_2\pi) + \frac{2}{|J_1|}\sum_{j\in J_1}(x_j - 2x_2\sin(2\pi x_1 + \frac{j\pi}{n}))^2 \\
f_2(X) &= \cos(0.5x_1\pi)\sin(0.5x_2\pi) + \frac{2}{|J_2|}\sum_{j\in J_2}(x_j - 2x_2\sin(2\pi x_1 + \frac{j\pi}{n}))^2 \\
f_3(X) &= \sin(0.5x_1\pi) + \frac{2}{|J_3|}\sum_{j\in J_3}(x_j - 2x_2\sin(2\pi x_1 + \frac{j\pi}{n}))^2 \\
&\quad J_1 = \{j|3 \le j \le n, \text{ and } j-1 \text{ is a multiplication of } 3\} \\
&\quad J_2 = \{j|3 \le j \le n, \text{ and } j-2 \text{ is a multiplication of } 3\} \\
&\quad J_3 = \{j|3 \le j \le n, \text{ and } j \text{ is a multiplication of } 3\} \\
&\quad n = 30; \quad [0,1]^2 \times [-2,2]^{n-2}
\end{aligned}$$

Fun11

$$\begin{aligned}
f_1(X) &= 0.5[\max\{0, (1+\varepsilon)(1-4(2x_1-1)^2)\} + 2x_1]x_2 \\
&\quad + \frac{2}{|J_1|}\sum_{j\in J_1}(x_j - 2x_2\sin(2\pi x_1 + \frac{j\pi}{n}))^2 \\
f_2(X) &= 0.5[\max\{0, (1+\varepsilon)(1-4(2x_1-1)^2)\} - 2x_1 + 2]x_2 \\
&\quad + \frac{2}{|J_2|}\sum_{j\in J_2}(x_j - 2x_2\sin(2\pi x_1 + \frac{j\pi}{n}))^2 \\
f_3(X) &= 1 - x_2 + \frac{2}{|J_2|}\sum_{j\in J_2}(x_j - 2x_2\sin(2\pi x_1 + \frac{j\pi}{n}))^2 \\
&\quad J_1 = \{j|3 \le j \le n, \text{ and } j-1 \text{ is a multiplication of } 3\} \\
&\quad J_2 = \{j|3 \le j \le n, \text{ and } j-2 \text{ is a multiplication of } 3\} \\
&\quad J_3 = \{j|3 \le j \le n, \text{ and } j \text{ is a multiplication of } 3\} \\
&\quad \varepsilon = 0.1; \quad [0,1]^2 \times [-2,2]^{n-2}
\end{aligned}$$

Fun12

$$\begin{aligned} f_1(X) &= (1+g(x))x_1x_2 \\ f_2(X) &= 1+g(x))x_1(1-x_2) \\ f_3(X) &= (1+g(x))(1-x_1) \\ & g(x) = 100(n-2)+100\sum_{i=3}^{n}\{(x_i-0.5)^2-\cos[20\pi(x_i-0.5)]\} \\ & n=10; \quad [0,1]^n \end{aligned}$$

Fun13

$$\begin{aligned} f_1(X) &= (1+g(x))\cos(\frac{x_1\pi}{2})\cos(\frac{x_2\pi}{2}) \\ f_2(X) &= (1+g(x))\cos(\frac{x_1\pi}{2})\sin(\frac{x_2\pi}{2}) \\ f_3(X) &= ((1+g(x))\sin(\frac{x_1\pi}{2}) \\ & g(x) = \sum_{i=3}^{n} x_i^2 \\ & [0,1]^2 \times [-1,1]^{n-2} \end{aligned}$$

Appendix B

Notations in Thesis

B.1 Notations in PSO and LA

N	dimensionality of the search space
n_{p}	number of particles in PSO
$\mathbf{Z}_i$	position vector of particle i, and $\mathbf{Z}_i$=(z_{i1},z_{i2},. . . ,z_{ij},. . . ,z_{iN})
$\mathbf{V}_i$	velocity vector of particle i, and $\mathbf{V}_i$=(v_{i1},v_{i2}, . . . ,v_{ij}, . . . ,v_{iN})
$\mathbf{P}_{\mathrm{l}i}$	the best position found by particle i, and $\mathbf{P}_{\mathrm{l}i}$=($p_{\mathrm{l}i1}$,$p_{\mathrm{l}i2}$,. . . ,$p_{\mathrm{l}ij}$,. . . ,$p_{\mathrm{l}iN}$)
$\mathbf{P}_{\mathrm{g}}$	the best position found in the swarm, and $\mathbf{P}_{\mathrm{g}}$=($p_{\mathrm{g}1}$,$p_{\mathrm{g}2}$,. . . ,$p_{\mathrm{g}j}$,. . . ,$p_{\mathrm{g}N}$)
n	iteration number (PSO), or time reference (LA)
w	inertia weight used in PSO
ζ_1, ζ_2	random numbers used in PSO
c_{f1}, c_{f2}	acceleration factors used in PSO
X	state (CALA) or context vector (GLA)
XN	number of states
$\underline{X}$	set of states (CALA), or set of context vectors (GLA)
a	action
$\underline{A}$	set of actions
aN	number of actions

r	reinforcement
$\underline{R}$	set of reinforcements
rN	number of reinforcements
$\tilde{\alpha}, \tilde{\beta}$	real numbers
p_i	probability of choosing action a_i
$\underline{P}$	set of action probabilities
$\mathcal{F}(\cdot,\cdot)$	transition function
$\mathcal{G}(\cdot)$	output function
$\mathcal{T}$	reinforcement scheme
$\underline{C}$	penalty probability
$\tilde{\lambda}$	step size of learning
F_i	reward probability of action a_i, or objective of learning
μ	mean of probability distribution
σ	standard deviation of probability distribution
a^*	the optimal action, or the optimal solution
I	index of the optimal action
σ_L	lower bound of σ
C_{P}	positive constant used in CALA
g	probability generating function
$\underline{S}$	internal state, $\underline{S} = (\bar{S}_1^T \ldots \bar{S}_{aN}^T)^T$
$\bar{S}_i$	the i^{th} vector in $\underline{S}$
L, K	constants
J	positive integer

B.2 Notations in FOLA and MOLA

N	number of automata, or dimensionality of the search domain
χ_i	set of possible states on the ith dimension
x_i	dimensional state on the ith dimension
$[x_{\min,i},\ x_{\max,i}]$	search range of dimension i
A_i	set of possible actions on dimension i
l	reference number of path: left path if $l = 1$; right path if $l = 2$
η	step length
r	a reinforcement signal
p_l	probability of selecting path l
$\mathfrak{T}$	reinforcement scheme
$F(\cdot)$	objective function
X	solution, or state
D	number of cells on one dimension
$c_{i,j}$	the j^{th} cell on dimension i
$w_{\text{c},i}$	the width of the cell in dimension i
$V(x_i)\vert_{x_i \in c_{i,j}}$	cell value of $c_{i,j}$
L_l	path value of path l
λ_1, λ_2, α	weights
τ	temperature
ξ	number of cells between the current cell and the selected cell
ζ	random number
k	constant number
X_{best}	the latest best solution
N_{femax}	the maximum number of objective function evaluations
I_{emax}	number of iterations in an episode

C	covariance matrix
c_{cov}	learning rate of the covariance matrix
σ	step-size
O, M, f_{bias}	shift, rotation and bias
F_{min}	the minimum value of objective function
f_i	the i^{th} objective function in MOLA
m_{f}	number of objectives
$\mathbb{L}$	elite list
B	set of the solutions which were stored in the elite list, but currently dominated by X_{best}
F^*	target objective vector
X_{lbest}	the latest local best solution
M	number of weight vectors
W	weight vector, and $\mathbf{W}^i = [w_1, w_2, \cdots, w_{\mathrm{mf}}]$
$D(i)$	set of indexes
X^i_{sub}	subsolution
z^*	reference point
$f_{j,\mathrm{range}}$	estimated range of f_j
$\mathrm{Nei}(X^i_{\mathrm{sub}})$	set of neighboring solutions of X^i_{sub}
I_1, I_2	indexes
ψ_1, ψ_2	weights
HV	hypervolume
$\tilde{D}$	distance metric
$\triangle$	diversity metric

B.3 Notations in Power Systems

X	control variables
U	dependent variables
P_{G_i}	injected active power at bus i (p.u.)
V_{G}	voltage at generator buses (p.u.)
T_i	tap position at transformer i
V_i	voltage magnitude at bus i (p.u.)
Q_{C_i}	reactive power source installation at bus i (p.u.)
N_{G}	set of numbers of generator buses
N_{T}	set of numbers of transformer branches
N_{C}	set of numbers of possible reactive power source installation bus
V_{L}	voltage at load buses (p.u.)
Q_{D_i}	demanded reactive at bus i (p.u.)
Q_{G_i}	injected reactive power at bus i (p.u.)
S_k	apparent power flow in branch k (p.u.)
N_{E}	set of numbers of network branches
N_{L}	set of numbers of load buses
N_i	total number of buses adjacent to bus i, including bus i
θ_{ij}	voltage angle difference between bus i and j (rad)
B_{ij}	transfer susceptance between bus i and j (p.u.)
G_{ij}	transfer conductance between bus i and j (p.u.)
N_0	total number of total buses excluding slack bus
N_{PQ}	total number of PQ buses
λ_{G_i}	penalty factors of the reactive power on the i^{th} bus
$N_{\mathrm{G}}^{\mathrm{lim}}$	set of numbers of generator buses on which injected reactive power outside limits
F	objective function
$\tilde{f}_i$	fuel cost of the i^{th} bus

a_i, b_i, c_i	fuel cost coefficients
ω	weighting factor
L_i	stability indicators at bus i
$L_{\max}$	global risk indicator
Y_{jj}^{+}	transformed admittance
S_j^{+}	transformed power
$\tilde{x}_m$	excitation reactance
$\tilde{x}_1$	stator reactance
$\tilde{x}_2$	rotor reactance
$\tilde{r}_2$	rotor resistance
$\tilde{r}_1$	stator resistance
$\tilde{s}$	slip of the asynchronous wind generator
V_{WG}	wind generator voltage
P_{WG}	active power of wind generator
Q_{WG}	reactive power of wind generator
N_{FG}	set of numbers of buses incorporated with fuel-derived power generators
h	price penalty factor
E_i	cost of pollutant emission of the ith generator
ρ	air density
A_{blade}	area swept by the turbine blades
R_{blade}	radius of turbine rotor blades
C_p	turbine performance coefficient
v_{w}	wind speed
$\alpha_{\mathrm{E}i}$, $\beta_{\mathrm{E}i}$, $\gamma_{\mathrm{E}i}$	coefficients of the ith generator's emission characteristics
$\zeta_{\mathrm{E}i}$, $\lambda_{\mathrm{E}i}$	coefficients of the ith generator's emission characteristics
P_{D_i}	demanded active power at bus i (p.u.)
$\tilde{B}_j$	benefit (or bid) function of customer j
$\tilde{C}_i$	cost (or bid) function of generator i

B.4 List of Abbreviations and Notations

Abbreviations

LA	Learning Automata
EAs	Evolutionary Algorithms
GA	Genetic Algorithm
EP	Evolutionary Programming
GP	Genetic Programming
ES	Evolutionary Strategy
FEP	Fast Evolutionary Programming
PSO	Particle Swarm Optimisation
CLPSO	Comprehensive Learning Particle Swarm Optimiser
ACO	Ant Colony Optimisation
GSO	Group Search Optimiser
FALA	Finite Action-set Learning Automata
CALA	Continuous Action Learning Automaton
GLA	Generalized Learning Automaton
PLA	Parameterized Learning Automaton
FOLA	Function Optimisation by Learning Automata
RLEP	Evolutionary Programming based on Reinforcement Learning
MSEP	Mixed Strategy Evolutionary Programming
CEP	Conventional Evolutionary Programming
CES	Conventional Evolution Strategies
FES	Fast Evolution Strategies
CPSO	Cooperative Particle Swarm Optimisation
FEs	Function Evaluations
GS-SOMA	Generalized Surrogate Single-Objective Memetic Algorithm
OLPSO	Orthogonal Learning Particle Swarm Optimisation
SOPEN	Self-Organizing Potential Field Network
SamACO	Sampling Ant Colony Optimisation

MOLA	Multi-objective Optimisation by Learning Automata
PGS	Pareto Global Search
PLS	Pareto Local Search
s.a.s	summary attainment surface
MOEA/D	Multi-Objective Evolutionary Algorithm based on Decomposition
MOPSO	Multi-Objective Particle Swarm Optimiser
MOGA	Multi-Objective Genetic Algorithm
NSGA-II	Fast Non-dominated Sorting Genetic Algorithm
OPF	Optimal Power Flow
ISO	Independent System Operator

Notations

$\preceq, \ \succeq$	vector inequality
$\forall$	for all
$\exists$	exist
T	transpose
$'$	derivative
∂	partial derivative

References

[1] X. Yao, Y. Liu, and G. Liu. Evolutionary programming made faster. *IEEE Transaction on Evolutionary Computation*, 3(2):82–102, 1999.

[2] B. Huang, P. Fery, L. Xue, and Y. Wang. Seeking the pareto front for multiobjective spatial optimization problems. *International Journal of Geographical Informaion Science*, 22(5):507–526, 2008.

[3] H. Li and Q. Zhang. Multiobjective optimization problems with complicated pareto sets, MOEA/D and NSGA-II. *IEEE Transations on Evolutionary Computation*, 12(2):284–302, 2009.

[4] W. C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17, 1991.

[5] E. Bonabeau, M. Dorigo, and G. Theraulza. Inspiration for optimisation from social insect behaviour. *Nature*, 406:39–42, July 2000.

[6] A. Cassioli, D. D. Lorenzo, M. Locatelli, F. Schoen, and M. Sciandrone. Machine learning for global optimization. *Computational Optimization and Applications*, published online (2010).

[7] A. Konak, D. W. Coit, and A. E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91(9):992 – 1007, 2006.

[8] K. H. Borgwardt. *The Simplex Algorithm: A Probabilistic Analysis, Algorithms and Combinatorics*. Springer-Verlag, 1987.

[9] B. Gartner and J. Matousek. *Understanding and Using Linear Programming*. Springer, Berlin, German, 2006.

[10] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, NY, USA, 1970.

[11] M. Avriel. *Nonlinear Programming: Analysis and Methods*. Dover Publishing, Mineola, NY, USA, 2003.

[12] D. G. Bounds. New optimization methods from physics and biology. *Nature*, 329(6136):215–219, 1987.

[13] R. M. Brady. Optimization strategies gleaned from biological evolution. *Nature*, 317(6040):804–806, 1985.

[14] H. B. Dong, J. He, H. K. Huang, and W. Hou. Evolutionary programming using a mixed mutation strategy. *Information Sciences*, 177(1):312–327, 2007.

[15] H. X. Zhang and J. Lu. Adaptive evolutionary programming based on reinforcement learning. *Information Sciences*, 178(4):971–984, 2008.

[16] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.

[17] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, 1966.

[18] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIET Press, 1992.

[19] I. Rechenberg. *Evolution Strategie: Optimieriung Rechnischer Systemenach Prinzipien der Biologichen Evolution*. Frommann-Holzboog, Stuuugart, Germany, 1973.

[20] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[21] L. N. Xing, Y. W. Chen, and K. W. Yang. Multi-population interactive co-evolutionary algorithm for flexible job shop scheduling problems. *Computational Optimization and Applications*, published online (2009).

[22] M. Bessaou and P. Siarry. A genetic algorithm with real-value coding to optimize multimodal continuous functions. *Structural and Multidisciplinary Optimization*, 23:63–74, 2001.

[23] F. Herrera, M. Lozano, and J. L. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12(4):265–319, 1998.

[24] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, 1:69–93, 1991.

[25] P. Hajela and C. Y. Lin. Genetic search strategies in multicriterion optimal design. *Structural and Multidisciplinary Optimization*, 4:99–107, 1992.

[26] T. Murata and H. Ishibuchi. MOGA: multi-objective genetic algorithms. In *Proc. of IEEE International Conference on Evolutionary Computation*, 29 1995.

[27] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2, 1994.

[28] G. G. Yen and H. M. Lu. Dynamic multiobjective evolutionary algorithm: adaptive cell-based rank and density estimation. *IEEE Transactions on Evolutionary Computation*, 7(3):253 – 274, 2003.

[29] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transaction on Evolutionary Computation*, 6(2):182–197, 2002.

[30] E. Masazade, R. Rajagopalan, P. K. Varshney, C. K. Mohan, G. K. Sendur, and M. Keskinoz. A multiobjective optimization approach to obtain decision

thresholds for distributed detection in wireless sensor networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 40(2):444–457, 2010.

[31] A. Vincenti, M. R. Ahmadian, and P. Vannucci. BIANCA: a genetic algorithm to solve hard combinatorial optimisation problems in engineering. *Journal of Global Optimization*, 48(3):399–421, 2010.

[32] D. Powell and M. M. Skolnick. Using genetic algorithms in engineering design optimization with nonlinear constraints. In *Proc. of International Conference On Genetic Algorithms*, pages 424–431, 1993.

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

[34] D. B. Fogel. *System Identification through Simulated Evolution: A Machine Learning Approach to Modeling*. Ginn Press, 1991.

[35] M. S. Li. *Bacteria-inspired Algorithms and Their Applications to Power System Optimisation*. Ph.D. thesis, Liverpool University, Liverpool, UK, 2010.

[36] B. Y. Qu and P. N. Suganthan. Multi-objective evolutionary programming without non-domination sorting is up to twenty times faster. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 2934 –2939, Trondheim, May 2009.

[37] K. P. Anchor, J. B. Zydallis, G. H. Gunsch, and G. B. Lamont. Different multi-objective evolutionary programming approaches for detecting computer network attacks. In *Proc. of the 2nd international conference on Evolutionary multi-criterion optimization*, EMO'03, pages 707–721, Berlin, Heidelberg, 2003. Springer-Verlag.

[38] P. Venkatesh and K. Y. Lee. Multi-objective evolutionary programming for economic emission dispatch problem. In *Proc. of IEEE Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century*, pages 1–8, Pittsburgh, PA, 2008.

[39] J. R. Koza and R. Poli. A genetic programming tutorial, 2003.

[40] H. P. Schwefel. *Numerical Optimization of Computer Models*. Wiley, Chichester, 1981.

[41] T. Back. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK, 1996.

[42] T. Back, F. Hoffmeister, and H. P. Schwefel. A survey of evolution strategies. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991.

[43] Z. Song and A. Kusiak. Multiobjective optimization of temporal processes. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 40(3):845–856, 2010.

[44] J. Kennedy, R. C. Eberhart, and Y. H. Shi. *Swarm Intelligence*. Morgan Kaufmann Publishers, 2001.

[45] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pages 1942–1948, IEEE Press, Piscataway, NJ, 1995.

[46] F. V. D. Bergh and A. P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information Sciences*, 176(8):937–971, 2006.

[47] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *IEEE World Congress on Computational Intelligence, IEEE International Conference on Evolutionary Computation*, IEEE Press, 1998.

[48] M. Clerc and J. Kenney. The particle swarm - explosion, stability, and convergence in a multimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[49] S. He, Q. H. Wu, J. Y. Wen, J. R. Saunders, and R. C. Paton. A particle swarm optimizer with passive congregation. *BioSystems*, 78:135–147, 2004.

[50] F. Bergh and A. P. Engelbrecht. A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):225–239, 2004.

[51] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar. Comprehensive learning particle swarm optimiser for global optimisation of multimodal functions. *IEEE Transactions on Evolutionary Compytation*, 10(3):281–295, 2006.

[52] A. Colorni, M. Dorigo, and V. Maniezzo. *Distributed Optimization by Ant Colonies*. Elsevier Publishing, Paris, France, 1991.

[53] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence*. Oxford University Press, Oxford, UK, 1999.

[54] V. Maniezzo, L. M. Gambardella, and F. D. Luigi. Ant colony optimization. In *Optimization Techniques in Engineering*, pages 101–117. Springer-Verlag, Addison-Wesley, 2004.

[55] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics–Part B*, 26(1):29–41, 1996.

[56] S. He, Q. H. Wu, and J. R. Saunders. Group search optimizer - an optimization algorithm inspired by animal searching behavior. *IEEE Transaction on Evolutionary Computation*, 13(5):973–990, 2008.

[57] J. W. Bell. *Searching Behaviour - The Behavioural Ecology of Finding Resources,* Chapman and Hall Animal Behaviour Series. Chapman and Hall, 1990.

[58] C. W. Clark and M. Mangel. Foraging and flocking strategies: information in an uncertain environment. *The American Naturalist*, 123(5):626–641, 1984.

[59] C. J. Barnard and R. M. Sibly. Producers and scroungers: a general model and its application to captive flocks of house sparrows. *Animal Behaviour*, 29:543–550, 1981.

[60] Q. H. Wu, Z. Lu, M. S. Li, and T. Y. Ji. Optimal placement of facts devices by a group search optimizer with multiple producers. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 1033–1039, 2008.

[61] R. R. Bush and F. Mosteller. *Stochastic Models for Learning*. Wiley, New York, 1958.

[62] G. H. Bower R. C. Atkinson and E. J. Crothers. *An Introduction to Mathematical Learning Theory*. Wiley, New York, 1965.

[63] M. L. Tsetlin. *Automaton Theory and Modeling of Biological Systems, vol. 102 in Mathematics in Science and Engineering*. Academic Press, New York, 1973.

[64] Q. H. Wu. Reinforcement learning control using interconnected learning automata. *International Journal of Control*, 62(1):1–16, 1995.

[65] H. E. Garcia, A. Ray, and R. M. Edwards. Reconfigurable control of power plants using learning automata. *IEEE Control Systems Magazine*, 11(1):85–922, 1991.

[66] C. Marsh, T. J. Gordon, and Q. H. Wu. Application of learning automata to controller design in slow-active automobile suspensions. *Vehicle System Dynamics*, 24(8):597–616, 1995.

[67] P. S. Sastry, G. D. Nagendra, and N. Manwani. A team of continuous-action learning automata for noise-tolerant learning of half-spaces. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 40(1):19–28, 2010.

[68] H. Beigy and M. R. Meybodi. Stochastic optimization using continuous action-set learning automata. *Scientia Iranica*, 12(1):14–25, 2005.

[69] A. S. Poznyak and K. Najim. *Learning Automata and Stochastic Optimization*. Springer, 1997.

[70] M. N. Howell, T. J. Gordon, and F. V. Brandao. Genetic learning automata for function optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 32(6):804–815, 2002.

[71] X. Zeng and Z. Liu. A learning automata based algorithm for optimization of continuous complex functions. *Information Sciences*, 174(3-4):165 – 175, 2005.

[72] M. R. Aghaebrahimi, S. H. Zahiri, and M. Amiri. *A New Method for Multiobjective Optimization Based on Learning Automata*. World Academy of Science, Engineering and Technology, 2009.

[73] M. A. L. Thathachar and P. S. Sastry. *Networks of Learning Automata: Techniques for Online Stochstic Optimization*. Kluwer Academic Publishers, 2004.

[74] K. S. Narendra and M. A. L. Thathachar. *Learning Automata: An Introduction*. Prentice Hall, Englewood Cliffs, 1989.

[75] K. S. Narendra and M. A. L. Thathachar. Learning automata - a survey. *IEEE Transactions on Systems Man and Cybernetics*, (4):323–334, 1974.

[76] R. Viswanathan and K. S. Narendra. Expedient and optimal variable-structure stochastic automata. Technical report CT-31, Yale University, New Haven, 1970.

[77] S. Lakshmivarahan and M. A. L. Thathachar. Optimal non-linear reinforcement schemes for stochastic automata. *Information Sciences*, 4(2):121 – 128, 1972.

[78] M. A. L. Thathachar and P. S. Sastry. Varieties of learning automata: an overview. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 32(6):711–722, 2002.

[79] H. Y. Kushner and G. G. Yin. *Stochastic Approximation Algorithms and Applications*. New York: Springer-Verlag, 1997.

[80] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[81] M. A. L. Thathachar and V. V. Phansalkar. Learning the global maximum with parameterized learning automata. *IEEE Transactions on Neural Networks*, 6(2):398–406, 1995.

[82] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, a Bradford Book, Cambridge, MA, 1998.

[83] J. S. Bridle. Training stochastic modal recognition algorithms as networks can lead to maximum mutual information estimates of parameters. In *Proc. of the Advances in Neural Information Processing Systems*, pages 211–217, Morgan Kaufmann, San Mateo, CA, 1989.

[84] Q. H. Wu and H. L. Liao. Function optimisation by learning automata. In *Proc. of the WCCI 2010 IEEE World Congress on Computational Intelligence*, pages 1–8, Barcelona, Spain, 2010.

[85] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. Technical report, Nanyang Technological University, Singapore and KanGAL Report Number 2005005, 2005.

[86] D. H. Wolpert and W. G. Macready. No free lunch theorems for search. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[87] A. Torn and A. Zilinskas. *Global optimisation, Volume 350 of Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1989.

[88] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, New York, 1995.

[89] H. P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.

[90] X. Yao and Y. Liu. *Fast Evolution Strategies, Evolutionary Programming VI, edited by P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart*. Springer, Berlin, 1997.

[91] C. Houck, J. Joines, and M. Kay. A genetic algorithm for function optimization: a matlab implementation. Technical report NCSU-IE-TR-95-09, Noth Carolina State University, Raleigh, NC, 1995.

[92] X. Yao and Y. Liu. Scaling up evolutionary programming algorithms. In *Proc. of the Seventh Annual Conference on Evolutionary Programming (EP98), Lecture Notes in Computer Science*, pages 103–112, Springer-Verlag, Berlin, 1998.

[93] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi. Scaling up fast evolutionary programming with cooperative coevolution. In *Proc. of the 2001 Congress on Evolutionary Computation*, pages 1101–1108, IEEE Press, Piscataway, NJ, USA, 2001.

[94] J. Biethahn and V. Nissen. *Evolutionary Algorithms in Management Applications*. Springer-Verlag, Berlin, 1995.

[95] N. Hansen. The CMA evolution strategy: a comparing review. In J. A. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, editors, *Towards a New Evolutionary Computation*, Studies in Fuzziness and Soft Computing, pages 75–102. Springer, 2006.

[96] D. C. Montgomery. *Statistical Quality Control*. Wiley and Sons, New York, 1996.

[97] D. Lim, Yaochu Jin, Yew Soon Ong, and B. Sendhoff. Generalizing surrogate-assisted evolutionary computation. *IEEE Transactions on Evolutionary Computation*, 14(3):329–355, 2010.

[98] Z. H. Zhan, J. Zhang, Y. Li, and Y. H. Shi. Orthogonal learning particle swarm optimization. *IEEE Transactions on Evolutionary Computation (to appear)*, 2011.

[99] L. Xu and T. W. S. Shing. Self-organizing potential field network: a new optimization algorithm. *IEEE Transactions on Neural Network*, 21:1482–1495, September 2010.

[100] X. M. Hu, J. Z., H. S. H. Chung, Y. Li, and O. Liu. Samaco: Variable sampling ant colony optimization algorithm for continuous optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 40(6):1555 –1566, 2010.

[101] B. T. Aharon. Characterization of pareto and lexicographic optimal solutions. *Lecture Notes in Economics and Mathematical Systems*, 177(1), 1980.

[102] D. A. V. Veldhuizen and G. B. Lamont. *Evolutionary Computation and Convergence to a Pareto Front*. Morgan Kaufmann, Stanford University, California, 1998.

[103] Q. F. Zhang and H. Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

[104] C. Gil, A. Marquez, R. Banos, M. G. Montoya, and J. Gomez. A hybrid method for solving multi-objective global optimization problems. *Journal of Global Optimization*, 38(2):265–281, 2007.

[105] G. Avigad and A. Moshaiov. Interactive evolutionary multiobjective search and optimization of set-based concepts. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 39(4):1013–1027, 2009.

[106] M. R. Sierra and C. A. C. Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International Journal of Computational Intelligence Research*, 2(3):287–308, 2006.

[107] N. R. Santiago L. V. S. Quintero and C. A. C. Coello. *Towards a More Efficient Multi-objective Particle Swarm Optimizer. In L. T. Bui and S. Alam, editors, Multi-Objective Optimization in Computational Intelligence: Theory and Practice.* IGI Global, 2008.

[108] C. M. Fonseca and P. J. Fleming. Multiobjective genetic algorithms made easy: selection sharing andmating restriction. In *Proc. of the First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 45–52, 1995.

[109] A. Osyczka and S. Kundu. A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm. *Structural and Multidisciplinary Optimization*, 10:94–99, 1995.

[110] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transaction on Evolutionary Computation*, 3(4):257–271, 1999.

[111] J. J. Durillo, A. J. Nebro, C. A. C. Coello, J. Garcia-Nieto, F. Luna, and E. Alba. A study of multi-objective metaheuristics when solving parameter scalable problems. *IEEE Transactions on Evolutionary Computation*, 14(4):618–635, 2010.

[112] C. M. Fonseca and P. J. Fleming. On the performance assessment and comparison of stochastic multiobjective optimizers. In *Lecture Notes in Computer Science, Proc. of the 4th International Conference on Parallel Problem Solving from Nature*, pages 584–593, Berlin, Germany, Springer-Verlag, 1996.

[113] http://dces.essex.ac.uk/staff/qzhang/moeacompetition09.htm. Technical report, 2009.

[114] W. Liu Q. Zhang and H. Li. The performance of a new version of moea/d on cec09 unconstrained mop test instances. Working report CES-491, School of CS & EE, University of Essex, 2009.

[115] M. S. Bond B. Scott J. A. Momoh, R. J. Koessler and D. Sun. Challenges to optimal power flow. *IEEE Transactions on Power Systems*, 12, 1997.

[116] N. G. Boulaxis, S. A. Papathanassiou, and M. P. Papadopoulos. Wind turbine effect on voltage profile of distribution network. *Renewable Energy*, 25, 2002.

[117] O. Alsac and B. Scott. Optimal load flow with steady state security. *IEEE Transactions on Power Apparatus and Systems*, 1974.

[118] M. H. Bottero, F. D. Galiana, and A. R. Fahmideh-Vojdani. Economic dispatch using the reduced hessian. *IEEE Transactions on Power Apparatus and Systems*, PAS-101:3679–3688, Oct. 1982.

[119] G. F. Reid and L. Hasdorf. Economic dispatch using quadratic programming. *IEEE Transactions on Power Apparatus and Systems*, 1973.

[120] B. Scott and E. Hobson. Power system security control calculation using linear programming. *IEEE Transactions on Power Apparatus and Systems*, 1978.

[121] J. A. Momoh and J. Z. Zhu. Improved interior point method for opf problems. *IEEE Trans on Power System*, 14:1114–1120, 1999.

[122] H. Wei, H. Sasaki, J. Kubokawa, and R. Yokoyama. An interior point nonlinear programming for optimal power flow using a novel structure. *IEEE Transactions on Power System*, 13, 1998.

[123] Q. H. Wu, Y. J. Cao, and J. Y. Wen. Optimal reactive power dispatch using an adaptive genetic algorithm. *International Journal of Electrical Power & Energy Systems*, 1998.

[124] B. Zhao, C. X. Guo, and Y. J. Cao. A multiagent-based particle swarm optimization approach for optimal reactive power dispatch. *IEEE Transactions on Power Systems*, 20(2):1070–1078, 2005.

[125] Lars Kroldrup. Gains in global wind capacity reported. Technical report, Green Inc., 2010.

[126] REN21. Renewables global status report. Technical report, 2009.

[127] Q. Ai and C. H. Gu. Economic operation of wind farm integrated system considering voltage stability. *Renewable Energy*, 34(3):608–614, 2009.

[128] W. J. Tang, M. S. Li, Q. H. Wu, and J. R. Saunders. Bacterial foraging algorithm for optimal power flow in dynamic environments. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 55(8):2433–2442, 2008.

[129] S. Brini, H. H. Abdallah, and A. Ouali. Economic dispatch for power system included wind and solar thermal energy. *Leonardo Journal of Sciences*, 8(14):204–220, 2009.

[130] U.S. Department of Energy. Final report on the August 14, 2003 blackout in the united states and canada. Technical report, U.S.-Canada Power System Outage Task Force, 2006.

[131] J. Carpentier. *Contribution to the Econimoc Dispatch Problem*, volume 8. Bull. Soc. Franc. Elect., 1962.

[132] Q. H. Wu and H. L. Liao. Function optimization by reinforcement learning for power system dispatch and voltage stability. In *Proc. of IEEE Power & Energy Society General Meeting*, pages 1–8, Minneapolis, USA, 2010.

[133] J. Grainger and W. Stevenson. *Power System Analysis*. McGraw-Hill, New York, 1994.

[134] S. He, E. Prempain, Q. H. Wu, J. Fitch, and S. Mann. An improved particle swarm optimization for optimal power flow. In *Proc. of IEEE 2004 International Conference on Power System Technology*, pages 21–24, The Pan Pacific, Singapore, F1628-2E2J(1-5), 2004.

[135] H. L. Liao and Q. H. Wu. Multi-objective optimization by reinforcement learning for power system dispatch and voltage stability. In *Proc. of IEEE PES Conference on Innovative Smart Grid Technologies Europe*, pages 1–8, Sweden, 2010.

[136] P. Kessel and H. Glavitch. Estimating the voltage stability of a power system. *IEEE Transaction on Power Delivery*, 3(1):346–354, 1986.

[137] K. M. Passino. http://www.ece.osu.edu/∼passino/icbook/ic_code.html. Technical report.

[138] B. Birge. Psot - a particle swarm optimization toolbox for use with matlab. In *IEEE international Conference on Swarm Intelligence Symposium*, pages 182–186, IEEE Press, 2003.

[139] Q. H. Wu, T. Y. Ji, M. S. Li, and Z. Lu. Group search optimizer with multiple producers for reactive power dispatch. *IEEE Transaction on Power Systems*, 2008.

[140] L. T. Ha and T. K. Saha. Investigation of power loss and voltage stability limits for large wind farm connections to a subtransmission network. In *Proc. of IEEE Power engineering society general meeting*, pages 2251–2256, Denver, CO, 2004.

[141] A. E. Feijdo and J. Cidris. Modeling of wind farms in the load flow analysis. *IEEE Transaction on Power Systems*, 15(1):110–115, 2000.

[142] Z. Chen and E. Spooner. Grid power quality with variable speed wind turbines. *IEEE Transactions on Energy Conversion*, 16(2):148–154, 2001.

[143] M. A. Abido. Environmental/economic power dispatch using multiobjective evolutionary algorithms. *IEEE Transactions on Power Systems*, 18(4):1529–1537, 2003.

[144] Y. M. Atwa, E. F. El-Saadany, M. M. A. Salama, and R. Seethapathy. Optimal renewable resources mix for distribution system energy loss minimization. *IEEE Transactions on Power Systems*, 25(1):360–370, 2010.

[145] P. Venkatesh, R. Gnanadass, and N. P. Padhy. Comparison and application of evolutionary programming techniques to combined economic emission dispatch with line flow constraints. *IEEE Transactions on Power Systems*, 18(2):688–697, 2003.

[146] E. Vittal, M. O'Malley, and A. Keane. Steady-state voltage stability analysis of power systems with high penetrations of wind. *IEEE Transactions on Power Systems*, 25(1):433–442, 2010.

[147] W. M. Lin and S. J. Chen. Bid-based dynamic economic dispatch with an efficient interior point algorithm. *International Journal of Electrical Power & Energy Systems*, 24(1):51–57, 2002.

[148] B. Zhao, C. X. Guo, and Y. J. Cao. Dynamic economic dispatch in electricity market using particle swarm optimization algorithm. In *Proc. of the Fifth World Congress on Intelligent Control and Automation*, volume 6, pages 5050 – 5054, 2004.

[149] X. Xia and A. M. Elaiw. Optimal dynamic economic dispatch of generation: A review. *Electric Power Systems Research*, 80(8):975 – 986, 2010.

[150] Q. H. Wu and Y. J. Cao. *Dispatching*. Encyclopaedia of Electrical and Electronics Engineering, edited by John G. Webster, John Wiley & Sons Inc., 1999.

[151] R. Bharathi, M. J. Kumar, D. Sunitha, and S. Premalatha. Optimization of combined economic and emission dispatch problem - a comparative study. In *Proc. of IPEC 2007 International Power Engineering Conference*, pages 134–139, Singapore, 2007.

[152] L. L. Freris. *Wind Energy Conversion System*. Prentice-Hall, Upper Saddle River, NJ, 1990.

[153] W. Zhou, Y. Peng, and H. Sun. Probabilistic wind power penetration of power system using nonlinear predictor-corrector primal-dual interior-point method. In *Electric Utility Deregulation and Restructuring and Power Technologies, 2008. DRPT 2008. Third International Conference on*, pages 2548–2552, 2008.

[154] J. Conroy and R. Watson. Aggregate modelling of wind farms containing full-converter wind turbine generators with permanent magnet synchronous

machines: transient stability studies. *IET Renewable Power Generation*, 3(1):39–52, 2009.

[155] G. Bathurst A. Shafiu, O. Anaya-Lara and N. Jenkins. Aggregated wind turbine models for power system dynamic studies. *Wind Engineering*, 30(3):171–186, 2006.

[156] V. Akhmatov. An aggregated model of a large wind farm with variable-speed wind turbines equipped with doubly-fed induction generators. *Wind Engineering*, 28(4):479–488, 2004.

[157] I. Patnaik. Wind as a renewable source of energy. *Technical Report*, 2009.

[158] X. Chen, H. S. Sun, J. Y. Wen, W. J. Lee, X. F. Yuan, N. H. Li, and L. Z. Yao. Integrating wind farm to the grid using hybrid multiterminal HVDC technology. *IEEE Transactions on Industry Applications*, 47(2):965–972, 2011.