Information Sciences 181 (2011) 2370–2391

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Dynamic multi-objective optimization based on membrane computing for control of time-varying unstable plants

Liang Huang [a], Il Hong Suh [b,*], Ajith Abraham [c]

[a] College of Information & Electronic Engineering, Zhejiang Gongshang University, PR China
[b] Intelligence and Communications for Robots Laboratory, Department of Computer Science and Engineering, College of Engineering, Hanyang University, Seoul, Republic of Korea
[c] Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence (SNIRE), Auburn, WA 98071, USA

ARTICLE INFO

Article history:
Received 24 June 2008
Received in revised form 21 December 2010
Accepted 31 December 2010
Available online 14 January 2011

Keywords:
Dynamic multi-objective optimization
Time-varying system
Membrane computing (P systems)
Membrane control strategy

ABSTRACT

Dynamic multi-objective optimization is a current hot topic. This paper discusses several issues that has not been reported in the static multi-objective optimization literature such as the loss of non-dominated solutions, the emergence of the false non-dominated solutions and the necessity for an online decision-making mechanism. Then, a dynamic multi-objective optimization algorithm is developed, which is inspired by membrane computing. A novel membrane control strategy is proposed in this article and is applied to the optimal control of a time-varying unstable plant. Experimental results clearly illustrate that the control strategy based on the dynamic multi-objective optimization algorithm is highly effective with a short rise time and a small overshoot.

© 2011 Elsevier Inc. All rights reserved.

1. Introduction

Dynamic optimization problems (DOPs) include dynamic single-objective optimization problems (DSOOPs) and dynamic multi-objective optimization problems (DMOOPs). Multi-objective optimization problems (MOOPs) are classified as static multi-objective optimization problems (SMOOPs) and DMOOPs. Most real-world optimization problems are dynamic, and they have multiple objectives; hence, they have attracted considerable interest as a frontier of science [2,12,34,36].

The literature on SMOOPs and DSOOPs has been greatly expanding with evolutionary algorithms being one of the most popular approaches [11,23,31,39] for solving practical problems. Multi-objective evolutionary algorithms can find a set of Pareto optimal solutions (POS) instead of a single optimal solution in a single simulation run [10,30,32,35]. Moreover, the uniformly distributed, near-optimal, and well-extended Pareto front illustrates the shape of the true Pareto optimal front (POF) as well as its extent. On the other hand, the objectives and/or environments change with time in many real world problems such as robot navigation, model identification and controller design. Therefore, many single objective problems have been studied, and many bionic approximate optimization algorithms have been proposed for different time-dependent problems [1,37].

In contrast, the intersection of MOOPs and DOPs, i.e., the DMOOPs, have not been explored extensively, as is evident in the available literature [9,20], despite the fact that many real-world problems naturally fall within the purview of DMOOPs. Jin and Sendhoff introduced an open scheme for generating test functions when shifting dynamically between static functions [17]. Farina et al. designed five dynamic functions with different complexities [9].

* Corresponding author. Tel.: +82 22220392.
E-mail address: ihsuh@hanyang.ac.kr (I.H. Suh).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2010.12.015

The current dynamic multi-objective test problems and optimization algorithms were developed from their static counterpoints [9]. As a result, two connotative principles have been accepted, despite a lack of investigation of their validity. The two assumptions are as follows. First, the dynamic *POS*(t) of a dynamic multi-objective optimization problem equals to the concatenation of a series of static Pareto-optimal solutions of its corresponding static problems in different time section. Second, the selection of the sole solution from the current POS does not affect the future dynamic multi-objective sub-problems. In other words, the current scenario is not related to the previous POS sets or the tradeoff solution. However, in real world applications, the two pre-suppositions seldom hold. This paper attempts to overcome some of these problems. Then, on the basis of analysis, a concrete dynamic multi-objective optimization algorithm is proposed, which is inspired by membrane computing.

Controller design is a classical filed of application for optimization algorithms [5,6,16,33]. Multiple performance criteria, such as maximum overshoot, settling time, rise time, phase margin, and gain margin, should be considered in order to design stable and powerful controllers. Many examples are available in the literature such as the optimization of H_2/H_∞ controller parameters [18,19] and the optimization of the fuzzy rule set for fuzzy controllers [3,13,14,22]. In these examples, optimizations were performed offline and the models of the plants or devices are not time-varying. In practical engineering problems, chaotic disturbances, randomness and complex nonlinear dynamic characteristics are often present. There are two types of approaches to the dynamic optimal control problems. One is the offline optimization of controller parameters by evaluating a controller in a number of real scenarios of the dynamic problems. Another popular approach is to search the optimal controller online. In the latter case, an optimization approach searches for the optimal controller within a given time span, during which the system is considered as stationary. The dynamic closed-loop system is regarded as a series of such stationary systems. This paper investigates the advantages and disadvantages of the two approaches, and integrates them to control a time-varying system.

Farina et al. discussed an adaptive control problem of a time-varying system as a special test case [9]. A time-dependent optimal controller is optimized online according to multiple objectives. However, a concrete dynamic controller was not realized. This paper investigated the test case introduced by Farina et al. [9] and proposed a novel control strategy based on dynamic multi-objective optimization.

This paper presents several notions on DMOOPs in Section 2. Several special test functions are presented in Section 3, and then some issues on DMOOPs are illustrated in Section 4. A novel dynamic multi-objective optimization algorithm based on membrane computing is presented in Section 5. The unstable dynamic time-varying plant is discussed in Section 6, and a control strategy based on dynamic multi-objective optimization approach is investigated in Section 7. Finally, the conclusions are provided in Section 8.

2. Dynamic multi-objective optimization problems

Without loss of generality, optimization problems are represented by minimization problems as:

Definition 1. A multi-objective optimization problem with constraints can be formatted as a vector function $\overline{f}$ that maps a set of N parameters (decision variables) to a set of M objectives [7,9,38].

$$\begin{aligned} &\min_{\overline{X}\in[\overline{L},\overline{U}]} \overline{f(\overline{X})} = \{f_1(\overline{X}), f_2(\overline{X}), \ldots, f_M(\overline{X})\};\\ &\text{s.t.} \quad \overline{g}(\overline{X}) \leqslant \overline{0}, \overline{h}(\overline{X}) = \overline{0};\\ &\qquad \overline{X} = (x_1, x_2, \ldots, x_N); \end{aligned} \tag{1}$$

where $\overline{X}$ is the decision vector; $\overline{L}$ and $\overline{U}$, the lower and upper boundaries of the search space; $\Omega_{\Re}$, the n-dimensional decision space; and $\overline{f}$, the objective vector with m dimensions. The Pareto-optimal solutions of a multi-objective optimization problem consist of all the decision parameters for which the corresponding objective vectors cannot be improved in any dimension without degradation in another.

Definition 2. As a natural extension, a dynamic multi-objective optimization problem can be generally described as a vector function with a time variable, while one or more components are made time-dependent in a static multi-objective optimization problem.

$$\begin{aligned} &\min_{\overline{X(t)}\in[\overline{L(t)},\overline{U(t)}]} \overline{f(\overline{X(t)},t)} = \{f_1(\overline{X(t)},t),\ f_2(\overline{X(t)},t), \ldots,\ f_{m(t)}(\overline{X(t)},t)\};\\ &\text{s.t.} \quad \overline{g(\overline{X(t)},t)} \leqslant \overline{0}, \overline{h(\overline{X(t)},t)} = \overline{0};\\ &\qquad \overline{X(t)} = (x_1(t), x_2(t), \ldots, x_{n(t)}(t)); \end{aligned} \tag{2}$$

where $t \in R$ is the time variable; $\overline{X(t)} = (x_1(t), x_2(t), \ldots, x_{n(t)}(t))$, the decision vector; $\Omega_{\Re}(t) = \{\overline{X(t)} | \overline{X(t)} \in [L(t), U(t)], \overline{g(\overline{X(t)},t)} \leqslant \overline{0}, \overline{h(\overline{X(t)},t)} = \overline{0}\}$, the decision vector space; $\overline{L(t)}$, the time-varying lower boundary; and $\overline{U(t)}$, the dynamic upper boundary of the search space. The time-dependent equality and inequality constraints are represented by $\overline{g(\overline{X(t)},t)}$ and

$\overline{h(\overline{X(t)}, t)}$, respectively. The dimension of the decision vector $n(t)$ and the number of objectives $m(t)$ are both time-varying. In summary, if any parameter of a MOOP changes, the problem turns into a DMOOP. This definition stresses that the parameters $\overline{X(t)}, \Omega_{\Re}(t), n(t)$ and $m(t)$ can be dynamic in many practical problems.

Definition 3. In the minimization case, a solution $\overline{U(t)}$ dominates $\overline{V(t)}$ at time t_p, if and only if, in the entire time range, there are

$$\begin{cases} \exists t_p \in [0,\infty); \\ \forall i,\ u_i(t_p) \leqslant v_i(t_p); \\ \exists j;\ u_j(t_p) < v_j(t_p); \\ i \neq j; \\ i,j \in \{1,2,\ldots,m\}; \\ \overline{U(t)} = (u_1(t), u_2(t), \ldots, u_n(t)); \\ \overline{V(t)} = (v_1(t), v_2(t), \ldots, v_m(t)); \\ n \geqslant m; \end{cases} \tag{3}$$

Definition 4. Vector $\overline{U(t)}$ dominates vector $\overline{V(t)}$ if $\overline{U(t)}$ dominates $\overline{V(t)}$ at any fixed time during the entire time range. This is described as in Eq. (4):

$$\begin{cases} \forall i,\ u_i(t) \leqslant v_i(t); \\ \exists j;\ \exists t_p \in [0,\infty);\ u_j(t_p) < v_j(t_p); \\ i \neq j;\ i,j \in \{1,2,\ldots,m\}; \\ \overline{U(t)} = (u_1(t), u_2(t), \ldots, u_n(t)); \\ \overline{V(t)} = (v_1(t), v_2(t), \ldots, v_m(t)); \\ n \geqslant m; \end{cases} \tag{4}$$

Definition 5. The Pareto-optimal solution is represented by $\overline{X(t)} \in \Omega_{\Re}(t)$ if $\overline{U(t)} = \overline{f(\overline{X(t)}, t)}$ is non-dominated in the entire time range by any $\overline{V(t)} = \overline{f(\overline{Y(t)}, t)},\ \forall \overline{Y(t)} \in \Omega_{\Re}(t)$. The set of all dynamic Pareto optimal solutions is represented by $POS(t)$. Their values are the dynamic Pareto optimal front, and they are represented by $POF(t)$. At a fixed time t_p, their corresponding values are $\overline{X(t_p)}$, $POS(t_p)$, and $POF(t_p)$.

Definition 6. If the time is fixed at t_p, a dynamic problem in Definition 2 can be transformed into the corresponding static problem in Definition 1. In the static problem, the subscript s is adopted to distinguish it from its counterparts in the dynamic problem. $\overline{X_s(t_P)} \in \Omega_{s\Re}(t_p)$ is a Pareto-optimal solution of a static optimization problem f_s at the fixed time t_p if $\overline{U(t)} = \overline{f_s(\overline{X(t_p)}, t_p)}$ is non-dominated by any vector $\overline{V(t)} = \overline{f_s(\overline{Y(t_p)}, t_p)},\ \forall \overline{Y(t_p)} \in \Omega_{s_{\Re}}(t_p)$. The Pareto optimal solution set and the Pareto optimal front at time t_p are represented by $POS_s(t_p)$ and $POF_s(t_p)$, respectively. If t_p is discretionary, $POS_s(t_p)$ and $POF_s(t_p)$ might be integrated as two new sets $POS_s(t)$ and $POF_s(t)$, respectively, over the entire time axis.

3. Test functions

A new algorithms should be detected and analyzed by using test functions. However, there is a lack of test functions for dynamic multi-objective optimization. Farina et al. have sorted the DMOOPs into four types according to the changes in the decision space or objective space. In addition, they designed the test functions according to different classes [7–9,17]. In this study, other scenarios are taken into consideration as a supplement.

3.1. Time dependent dimension in decision space

$$\mathrm{T1}: \begin{cases} f_1(\overline{X}, t) = \sum_{i=1}^{d1(t)} (x_i^2 - 10\cos(2\pi x_i) + 10); \\ f_2(\overline{X}, t) = (x_1 - 1)^2 + \sum_{i=2}^{d2(t)} (x_i^2 - x_{i-1})^2; \\ d_1(t) = \lfloor n|\sin(t)| \rfloor; \\ d_2(t) = \lfloor n|\cos^3(2t)| \rfloor; \\ t = \frac{1}{n_t}\left\lfloor \frac{\tau}{\tau_T} \right\rfloor. \end{cases} \tag{5}$$

Eq. (5) is the scenario, where the dimension changes in decision space. The two objectives obtain global minimization 0 at $x_i = 0$ and $x_i = 1$, respectively. Similar to test function FDA1 [15], the number of generations counter is τ. The number of generations is τ_T during which t remains fixed, and n_t is the number of distinct steps in t. The valuation can be assumed as $n = 20$, $\tau_T = 5$, and $n_t = 10$. The changed dimensions are represented by two integer functions $d_1(t)$ and $d_2(t)$ with an upper limit n, as the equation given above. While their values increase simultaneously, the two objectives conflict with each other seriously. In practice, there are many cases where the objective functions have not changed, whereas the dimension of their decision space is time-varying. For example, in optimal control online, different control strategies with different numbers of parameters are adopted at different stages.

3.2. Time dependent dimension in objective space

If the number of objectives M in test function DTLZ is transformed into a time-varying function $m(t)$, the test function is transformed into Eq. (6). In this case, the dimension of the decision space and the optimal values of $n - M$ variables do not change; however the dimension of the objective space and the shape of $POF(t)$ change with time.

$$\mathrm{T2}: \begin{cases} \min. \quad f_1(x) = (1 + g(\overline{X_{II}})) \prod_{i=1}^{m(t)-1} \cos\left(\frac{\pi x_i}{2}\right); \\ \min. \quad f_k(x) = (1 + g(\overline{X_{II}})) \prod_{i=1}^{m(t)-k} \cos\left(\frac{\pi x_i}{2}\right) \\ \qquad \sin\left(\frac{\pi x_{m(t)-k+1}}{2}\right); \quad k = 2 : m(t) - 1; \\ \min. \quad f_{m(t)}(x) = (1 + g(\overline{X_{II}})) \sin(\frac{\pi x_1}{2}); \\ \min. \quad g(X_{II}) = \sum_{i=1}^{m(t)} (x_i - 0.5)^2; \\ \qquad m(t) = \lfloor M \times |\sin(0.5\pi t)| \rfloor, \quad t = \frac{1}{n_t} \left\lfloor \frac{\tau}{\tau_T} \right\rfloor; \\ \qquad x_i \in [0, 1], \quad for\ i = 1, 2, \ldots, n. \end{cases} \tag{6}$$

3.3. Current solutions depend on previous solutions

$$\mathrm{T3}: \begin{cases} f_1(x) = R(\overline{X}, t) \cos(\frac{\pi x_1}{2}); \\ f_2(x) = R(\overline{X}, t) \sin(\frac{\pi x_1}{2}); \\ \text{where}: \\ R(\overline{X}, t) = avgR(t-1) + G(\overline{X}, t); \\ avgR(t-1) = \frac{1}{P} \sum_{j}^{P} R_j(\overline{X}, t-1); \\ avgR(-1) = 1; \\ t = \left\lfloor \frac{\tau}{\tau_T} \right\rfloor; \\ G(\overline{X}, t) = \sum_{i=2}^{n} (x_i - avgR(t-1))^2; \\ x_1(t) \in [0, 1]; \\ x_i(t) \in [avgR(t) - 100, avgR(t) + 100]; \\ i = 2, \ldots, n. \end{cases} \tag{7}$$

This is a complex test function. Its true POF is a quadrant with constant radius 1. The true POS is also a constant. However, almost all algorithms will fail to find the true POF. If a slight error appears, the true radius will increase and the error will be cumulated to the radius of further optimization sub-problems. In fact, the resulting $POF(t)$ is a series of quadrants with increasing radii; the current radius depends on the previous optimization results. $G(\overline{X}, t)$ might be replaced by other single optimization functions to produce different test functions. The definition domain also changes with the previous computing result.

3.4. Current solutions depend on previous decision-making

$$\mathrm{T4}: \begin{cases} f_1(X,t) = \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i) + 10); \\ f_2(X,t) = (x_1 - r(t))^2 + \sum_{i=2}^{n}(x_i^2 - x_{i-1})^2; \\ t = \frac{1}{n_t}\left\lfloor \frac{\tau}{\tau_T} \right\rfloor. \end{cases} \tag{8}$$

In the test function, $POF(t)$ depends on the previous trade-off solution and the optimal solution set. The trade-off solution is $s_t(t)$, which is derived from the current solution set by the decision making mechanism. The average error of all its variables is represented by $r(t) = \frac{1}{n}\sum_{i=1}^{n}(x_i - 0)$. The function f_2 has a local minimum, $r(t) = 0$. If it appears, the two objectives will be consistent.

4. Optimization approaches and analysis

Although DMOOPs are the intersection of MOOPs and DOPs, the strategies and research works on MOOPs and DOPs are not identical; DMOOPs have their own characteristics. In this study, DMOOPs are classified as two types of problems, and they are investigated via given corresponding approximate optimization approaches. Since objectives receive more attention to than other factors, slow-change problems and fast change problems are investigated according to the rate of objective change with time in objective space.

4.1. Slow-change case

Irrespective of gradual or sudden changes with a comparatively long static state in objective functions, it is natural to consider them as stationary sub-functions in different time periods. In other words, this type of DMOOPs can be solved by using a stationary optimization algorithm several times within the corresponding time spans in which the objective functions are stationary [9].

In the stationary case, the criterion for a good algorithm is that it can quickly obtain its solution set as close as possible to the Pareto-optimal front. This is true in the dynamic case, and the speed of convergence is emphasized. In the worst case, the optimal front should be tracked or the transient Pareto-optimal solutions should be computed out before the objectives change significantly. If the entire time range is divided into n nonintersecting sections T_{si}, $i = 1,2,\ldots,n$, during which the objectives are assumed to be stationary, the DMOOPs can be divided into n static optimization problems (SMOOPs). The POS of the DMOOP can be approximately regarded as the optimal solution superimposition of some SMOOPs on different instants, if the time span $T_{si} \to 0$ and the subproblem at T_{si} does not depend on the previous subproblems at 0, $T_{s1},\ldots,T_{si-1}$.

If an algorithm needs a time interval T_g to complete one optimization iteration and τ iterations are allowed to track the Pareto-optimal front, the optimization time $T_o = T_g\tau$ must be shorter than T_s, otherwise, the algorithm fails to track the dynamic $POF_s(t)$. Fig. 1 illustrates this relationship. The superimposition of static objective $f_s(t)$ approximately describes the shape of the dynamic $f(t)$. $f_s(t)$ tracks $f(t)$ with a delay time that is equal to the optimization time T_o. Therefore, the static algorithms can be used without modification if $T_g \ll T_s$. If $T_g > T_s/2$, it is considered a fast-change problem and new methods should be found.

4.2. Fast-change case

Evolutionary algorithms provide solutions after one or more iterations because the solutions need time to evolve. Their optimal candidate solutions need time to be selected after evaluation and comparison. In other words, an evolutionary algorithm needs more than 2 iterations to improve its candidate solutions. Therefore, if $T_g > T_s/2$, i.e. $\tau = T_o/T_g < T_s/T_g < 2$, the dynamic problems should be transformed. If T_g is decreased or T_s is increased, the fast-change problem turns into a slow-change problem. The time span T_g is related to the algorithm itself and the computational power of the computer. The time span T_s is closely related to the objective function. The characteristics of an objective functions could be outlined

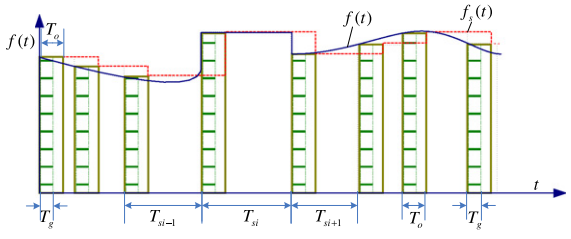

Fig. 1. Dynamic problem is converted into static problem.

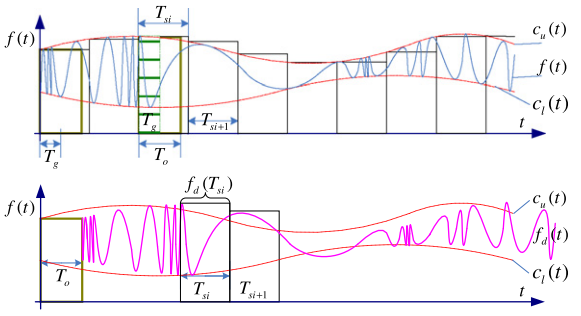

Fig. 2. Fast-varying problem.

by other functions, such as its $\|\cdot\|_1$, $\|\cdot\|_2$ and $\|\cdot\|_\infty$. If the new characteristic function changes slowly, the time span T_s maybe longer than the algorithm's T_g.

According to the new objective function, a fast-change optimization problem is divided into a series of subproblems like the slow-change optimization problem. The superimposition of these subproblem solutions is considered as the solution of the entire dynamic problem. The dynamic subproblems are dealt with as if they were static problems. However, the evaluation outcome of a subproblem may be dynamic according to the original objective as illustrated in Fig. 2 since the original objective changes dramatically during the term T_s. The comparison of the dynamic results could be executed according to Definitions 5 and 6. As an approximate strategy, this transfer should be further investigated in theory and in practice. In practice, adopting the characteristic functions as a new objective is useful. For instance, the rising time, overshoot, settling time, and $J = \int_0^\infty t|error(t)|dt$ are the often-used characteristic functions to describe the output curve of a control system.

4.3. Loss of nondominated solutions and the false nondominated solutions

The main idea of the approaches mentioned above is that the solutions of a dynamic multi-objective problem are approximated by the superposition of a series of static solutions over the entire time axis. In practice, the evolutionary algorithms fail to search out all solutions on the Pareto optimal front. If solutions are abundant, the set of solutions can describe the position, the shape, and other characteristics of the true Pareto-optimal front. In the dynamic optimization case, the situation is more complex. Evolutionary algorithms give some false global nondominated solutions, and even lose true nondominated solutions both in theory and in practice.

As depicted in Fig. 3, in the objective space, it has assumed that an optimization problem has four nondominated solutions on one objective image during the entire time range. These complete solutions are segmented according to each time span. An optimization algorithm gets a non-dominated solution $y_1(T_{si-1})$ at the stage T_{si-1}, obtains $y_3(T_{si})$ at the stage T_{si}, and searches out $y_3(T_{si+1})$ during T_{si+1}. However, the superimposition of the three nondominated solutions, $u(t)$ is not a feasible solution to the dynamic optimization problem. It is a false nondominated solution in the objective space. In other words, there is $POS(t) \neq POS_s(t)$ and $POF(t) \neq POF_s(t)$.

On the other hand, the algorithm discards a nondominated solution$y_2(t)$ in objective space because it never achieves the absolutely dominated state in any time span. However, it is a true nondominated solution during the entire time range according to Definition 4.

Difficulty arises because many problems depend on the results given or decisions made during the previous time span. As illustrated in Fig. 3, if $y_3(T_{si+1})$ is chosen as the sole solution at stage T_{si+1}, only v_2, v_3 are candidate solutions for the next stage and the subsequent solutions of $y_1(T_{si+1})$, v_1, v_4 are excluded. It is obviously in this case that the DMOOPs at this time cannot be replaced by the corresponding SMOOPs, which are obtained only by fixing the time of the original problems.

5. Algorithm design based on the dynamic multi-objective optimization

In order to solve the dynamic multi-objective optimization problems above, this paper introduces a novel dynamic multi-objective optimization algorithm inspired by P systems (DMOAP). P systems (membrane computing) are nondeterministic

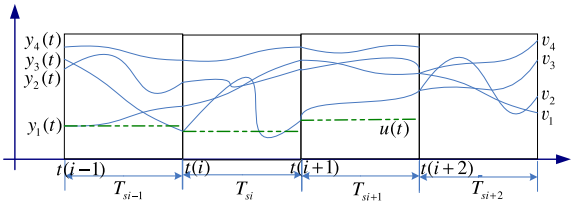

Fig. 3. Dynamic solutions and static solutions.

theoretical computing devices, which are abstracted from the structure and functioning of living cells and from the interactions of living cells in tissues or higher order biological structures [25,26,29]. Due to the P system's powerful computing capability [21,27] and its similarity to evolutionary computation, several corresponding optimization algorithms have been proposed that were inspired by P systems [4,15,24,28].

As discussed above, the entire dynamic $POF(t)$ and $POS(t)$ can be approximated by the superimposition of a series of corresponding static optimization results. Therefore, the task of an algorithm is designing and optimizing an equivalent static problem during each given term T_{si}. Since $POF(t_i)$ and $POS(t_i)$ are both stationary, static multi-objective algorithms could be

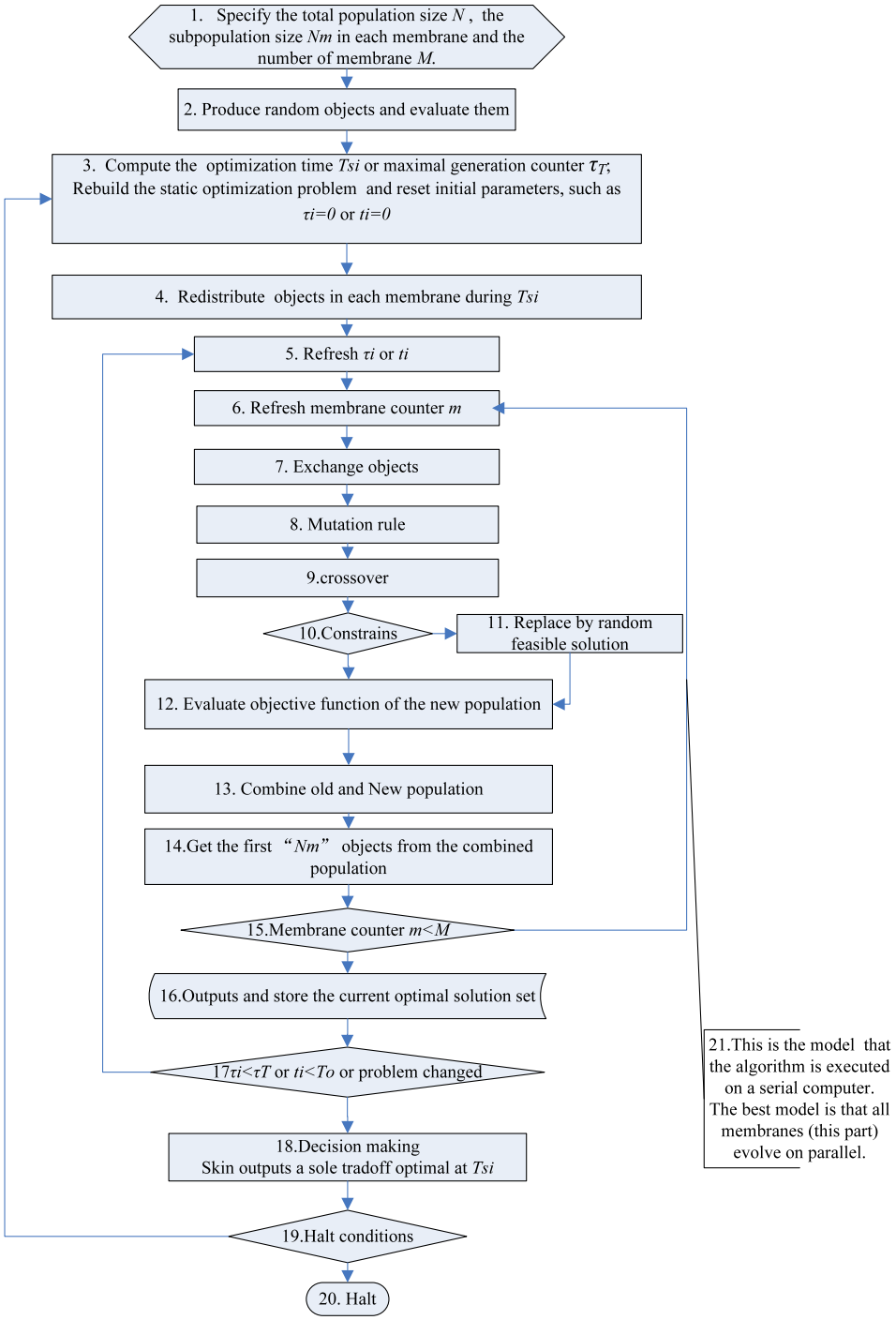

Fig. 4. DMOAP flowchart.

adopted as the nuclear procedure. When the current term T_{si} ends, a new equivalent $SMOOP(T_{si+1})$ is rebuilt during the time span T_{si+1} according to the *DMOOP* and the previous optimized results. Then, a static multi-objective optimization algorithm will restart with new initial conditions. The next $POF(t_{i+1})$ and $POS(t_{i+1})$ will then be calculated. The entire outline $POF(t_i)$ and $POS(t_i)$ will be outlined by the discrete sets of $POF(t_i)$ and $POS(t_i)$ at each time span T_{si}. A flowchart of this process is shown in Fig. 4 and detailed procedure is described below:

Box 1: In the initial stage, the systemic structure of the algorithm is chosen and the corresponding parameters are specified. As depicted in Fig. 5, DMOAP borrows a type of classical P system structure that consists of membranes. These membranes are grouped into $m + 1$ subsystems. m Sub-systems are single objective optimization subsystems that only optimize a corresponding objective. Another subsystem is the true multi-objective optimization sub-systems that optimizes all objectives synchronously. Each membrane has its own subpopulation and works like a single evolutionary algorithm. These membranes are contained within two special membranes that collect the resulting chromosomes from subsystems. In them, the chromosomes will not evolve and some inferior chromosomes will be removed. Membrane m_{mid} accepts the resulting chromosomes from all subsystems and sends the current Pareto optimal solution to the skin membrane. In effect, the skin membrane is a decision making table, which selects the sole trade-off solution for the current equivalent optimization sub-problem. In some special cases, it might pick up several trade-off solutions as the practical solutions. In the initial step, these parameters are evaluated, such as the number of objective functions M, the number of membranes in each subsystem m_i, the number of chromosomes in every membrane N_i and other parameters. Fig. 5 illustrates the structure of the algorithm. Its structure is divided into two SOOP subsystems and one MOOP subsystem. Each subsystem is divided into three membranes (regions). The number of chromosomes in each membrane is shown in Table 1.

Box 2: In this stage, objects (chromosomes) with given quantity are produced at random in the decision space. They are immediately evaluated according to each objective function. The chromosomes are coded by the connection of variables. They can be described as follows:

As shown in Fig. 6, a chromosome includes two genes: one gene presents the new candidate solutions, while the other describes the parameters of the equivalent optimization sub-problem when the current solution first appears. The evaluations of the candidate solution on different objectives compose the signal head. This is a similar process to that performed by a signal peptide in biology, which decides where the chromosome goes.

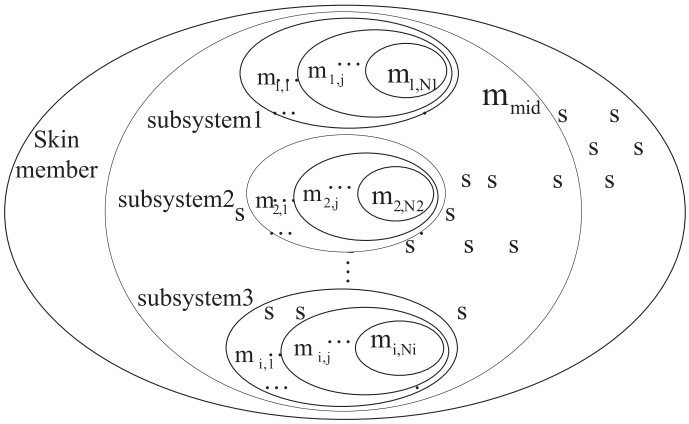

Fig. 5. Structure of DMOAP.

Table 1
Chromosome distribution.

Membrane label	$M_{1,1}$	$M_{1,2}$	$M_{1,3}$	$M_{2,1}$	$M_{2,2}$	$M_{2,3}$	$M_{3,1}$	$M_{3,2}$	$M_{3,3}$
Size of subpopulations (object set)	2	3	5	5	10	5	5	30	35

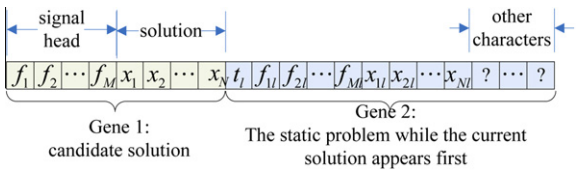

Fig. 6. Chromosome.

Box 3: The static sub-problem is redefined and the corresponding parameters are refreshed. The optimization time T_o or the maximal generations τ_T should be worked out according to T_s, during which the dynamic sub-problem is regarded as a stationary sub-problem. In test cases, the parameters might be given beforehand. In practice, it is difficult to predict the online optimization problem. This stage should be restarted by a stochastic interruption, which shows that the dynamic subproblem can be replaced by an new static equivalent problem. Therefore, the time span T_s is usually not constant. After the new static multi-objective optimization subproblem is rebuilt during the corresponding time span T_s, other corresponding parameters are reset, for example, the generation counter τ_i or the optimization timer t_τ. The decision space must also be refreshed.

Box 4: The objects in each membrane are redistributed according to their signal head at this stage. The new objects are produced randomly or come from the membrane m_{mid} that stored the previous $POS(t_f)$. Each candidate solution that obtains the best value on each objective is reserved. 60% of the total candidate solution set, $POS(t_f)$ is selected from previous solution set, $POS(t_f - 1)$ whose gene2 is close to the current static optimization subproblem.

There are advantages to adopting the previous solutions $POS(t_f - 1)$, though they might be dominated solutions in the current static problem. First, these solutions are better than random feasible solutions in most practical engineering cases. Second, it is possible that some solutions are non-dominated solutions during several different time spans. In short these solutions might improve the convergence and precision of an algorithm.

Box 5: Refresh the generation counter or optimization timer for the next generation of the operation.

Box 6: Refresh the membrane counter and prepare for the next membrane evolution if the algorithm is realized on a serial computer. In practice, each membrane should be implemented on its own computing device synchronously.

Box 7: The communication rule is applied. The membrane sends out some objects and adopts the objects that come from another membrane to form the new subpopulation. This is shown in Eq. (9):

$$\begin{bmatrix} {}_{i+1}[{}_i a_{better1}, a_{better2}, \ldots, a_{bettern}]_i \\ a_{worse1}, a_{worse2}, \ldots, a_{worsen} \end{bmatrix}_{i+1} \rightarrow \begin{bmatrix} {}_{i+1}[{}_i a_{worse1}, a_{worse2}, \ldots, a_{worsen}]_i \\ a_{better1}, a_{better2}, \ldots, a_{bettern} \end{bmatrix}_{i+1}, \tag{9}$$

Box 8: The offspring population is created by using the mutation rule according to in Eq. (10):

Chromosome/Object $X \rightarrow Y$

$$\begin{cases} X & x_1 & x_p & x_q & x_m \\ \Downarrow & & \xi_p & \xi_q & \\ Y & y_1 & y_p & y_q & y_m \end{cases} \tag{10}$$

where, $\xi = \eta_i \times \lambda$ is a random number, and $\lambda \subset [0,1]$ is a normal or uniformly distributed random number. η_i is the maximum of the parameter mutation. As shown in Table 2, their values are different in different membranes. Experiments show the performance of algorithm is best if the η_i of membrane $M_{j,i}$ is approximately equal to tenth of η_{i+1} of membrane $M_{j,i+1}$.

Box 9: Other operators of the evolutionary algorithm can be selected to improve the performance of the algorithm. In the following experiments, only crossover rule is explored.

Box 10: In this stage, the candidate solutions are verified according to the constraint conditions.

Box 11: If the constraints are not satisfied, the candidate solution is replaced by a random feasible solution.

Box 12: The new children population is evaluated.

Box 13: The new and old objects/chromosomes are combined and produce a new population with a doubled value of N_m.

Box 14: The best N_m nondominated chromosomes according to the domination relationship and niche strategy are compiled. The crowding distance is used at this stage [8].

Box 15: The next evolution is implemented until all membranes have been implemented.

Box 16: The current $POS(t_i)$ is sent to and stored in the membrane m_{mid}. In it, only nondominated solutions are preserved.

Box 17: If the given τ_T or T_{si} is not achieved, the algorithm will continue the optimization of the current $SMOOP(t_i)$. In practical engineering, the dynamic problem should be detected online. If a major change occurs or the current $SMOOP(t_i)$ has turned into another, an interruption must be carried out and a new $SMOOP(t_{i+1})$ should be rebuilt as soon as possible.

The time span T_s can be given in most test functions. However, it is the time span between two successive interruptions that represents a major change. The change should be detected online. There are two methods for change detection in different engineering scenarios. One is to detect dynamic factors described by the gene 2 in a chromosome. For example, if the only dynamic factor in an optimization problem is formulated as:

Table 2
Mutation parameters.

Membrane	$M_{1,1}$	$M_{1,2}$	$M_{1,3}$	$M_{2,1}$	$M_{2,2}$	$M_{2,3}$	$M_{3,1}$	$M_{3,2}$	$M_{3,3}$
η_i	0.2	0.02	0.001	0.05	0.01	0.001	0.1	0.05	0.005

$$f(t) = sign(t) = \begin{cases} 1 & t < t_c; \\ -1 & t \geqslant t_c; \end{cases} \tag{11}$$

The DMOOP is considered to be two sheer SMOOPs before and after t_c. If the switch time t_c is not already known, the value of the $f(t)$ should be detected online. After t_c, a major change will be detected because $f(t)$ jumps to 1 or the change $\Delta\|f(t)\|$ is larger than a selected value. Therefore, the detection unit sends an interruption. A new SMOOP should be rebuilt and the new POF and POS should be searched as soon as possible.

Another method is that a major change is judged according to the change of the objective functions [9].

$$\varepsilon(t) = \frac{\sum_{i=1}^{n} \frac{\|\overline{f_i(t)} - \overline{f_i(t-1)}\|}{\|\overline{f_i(t-1)}\|}}{n} \geqslant \varepsilon_G; \tag{12}$$

where, the number of chromosomes n is chosen randomly, and ε_G is the previously given value.

Box 18: If a substantial change occurs, the skin membrane will immediately create a tradeoff solution as the optimization result of the SMOOP during T_{si}. The trade-off solution is calculated from the current POS according to its selection rule.

Box 19: The termination criterion is examined. If this criterion is satisfied, the algorithm ends. Otherwise, the next static optimization will be regenerated according to the current state of the dynamic problem and a new search will begin.

6. Application in controller design for a time-varying unstable plant

Controller design is a good test case for new algorithms. The control of a time-varying unstable plant is a very challenging problem. As illustrated in Fig. 7, Farina et al. [6] discussed a dynamic controller optimization problem as a case of multi-objective optimization problem. However, they could not develop a concrete control strategy [9]. The test case is the control of a randomly varying plant by a proportional-integral-derivative (PID) controller. The transfer functions of the plant and controller are given as follows:

$$\begin{cases} G(s) = \frac{1.5}{50s^3 + a_2(t)s^2 + a_1(t)s + 1}; \\ C(s) = K_p(t) + K_i(t)\frac{1}{s} + K_d(t)s; \end{cases} \tag{13}$$

where $a_1(t)$ and $a_2(t)$ are time-varying parameters that simulate the aging or intrinsic random changes in system. In this case, these variables are given as follows:

$$\begin{cases} a_1(t) = 3 + 30f(t); \\ a_2(t) = 43 + 30f(t); \end{cases} \tag{14}$$

where function $f(t)$ is used to simulate different time-dependent scenarios, and is specifically given here as,

$$f(t) = \sin\left(\frac{\pi t}{18}\right). \tag{15}$$

Apparently, the plant has different models at different times. The change of plant structure should be considered in complicated scenarios. Therefore, a time-varying controller is necessary for the time-dependent plant in order to realize optimal control. In order to evaluate a controller or a control strategy, the rising time R, the maximum overshooting O, and the settling time ST are usually adopted as criteria in the time domain. If the plant of Eq. (13) is considered to be a series of problems in different small time segments, the three criteria are time-varying from one time segment to another, which are the short rising $R(t)$, the small maximum overshooting $O(t)$, and the settling time $ST(t)$. Therefore, the multi-objective optimization problem is formulated as follows:

$$\min.\ \{R(t), O(t), ST(t)\}. \tag{16}$$

In order to simplify the problem, Farina et al. [6] fixed the derivative coefficient in the PID controller to $K_d = 8.3317$. The other two coefficients, $K_i(t) \in (0.5, 5.0)$ and $K_p(t) \in (0.1, 1.0)$ could be adjusted in the given areas. Taking $R(t)$ and $O(t)$ as the optimization criteria, Farina discussed the set of Pareto optimal solutions for nine time steps where $f(t) = \sin(\frac{\pi t}{18})$.

$$\min_{(K_p(t), K_i(t)) \in (0.5, 5.0) \times (0.1, 1.0)} \{R(K_p(t), K_i(t)), O(K_p(t), K_i(t)), ST(K_p(t), K_i(t))\}. \tag{17}$$

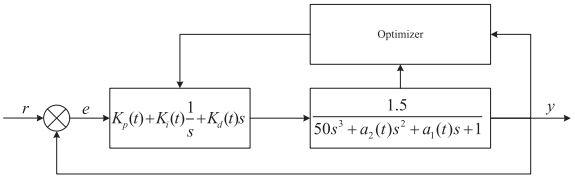

Fig. 7. Closed-loop control system.

As shown in Figs. 8–11, the shapes and values of $POF(t)$ and $POS(t)$ change sharply as time progresses. During a change cycle of $f(t)$, the distribution of nondominated solutions of POF also changes quickly. Thus, it is required that the dynamic optimization algorithm be sufficiently flexible to work out the approximately POF and POS as fast as possible.

The test case mentioned above is sufficiently close to the practical applications of multi-objective optimization problems. This case illustrates the basic idea and usefulness of multi-objective optimization methods. However, in the scope of control science, a concrete controller or control strategy has not yet been devised [9]. Future investigation should strive for the realization of ideal control.

The DMOAP adopted values of $N = 100$ and $\tau = 100$. The series of $POF_s(t)$ is obtained as depicted by Figs. 8–11. In the figures, the controllers that do not satisfy the following constraints were removed.

$$\begin{cases} T_r < 50; \\ \delta < 80; \\ T_s < 100. \end{cases} \tag{18}$$

7. Control strategy based on multi-objective optimization

In order to solve this difficult control problem and illustrate the application of multi-objective optimization methods, a control strategy for the test control problem mentioned above was worked out based on the multi-objective algorithm. Three hypotheses are given as follows:

(a) The change in the plant is unpredictable or a predictive tool has not been adopted, although this would be a useful strategy in practical engineering. In other words, the precise plant model's future state is unavailable.
(b) The current state is precise and the error and time delay of plant recognition can be ignored.
(c) The controller design is based on the Eq. (13). The same general procedure may be applied to a different control strategy, such as fuzzy control or H_∞, which is more practical for this type of plant.

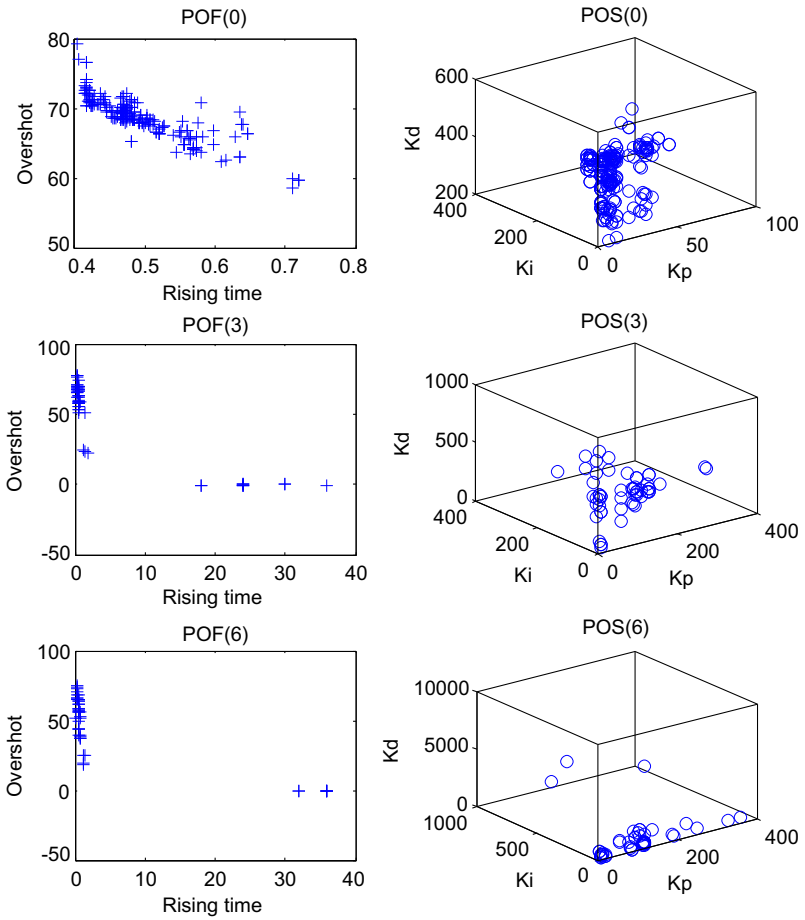

Fig. 8. POS of static problems ($t = 0, 3, 6$ s).

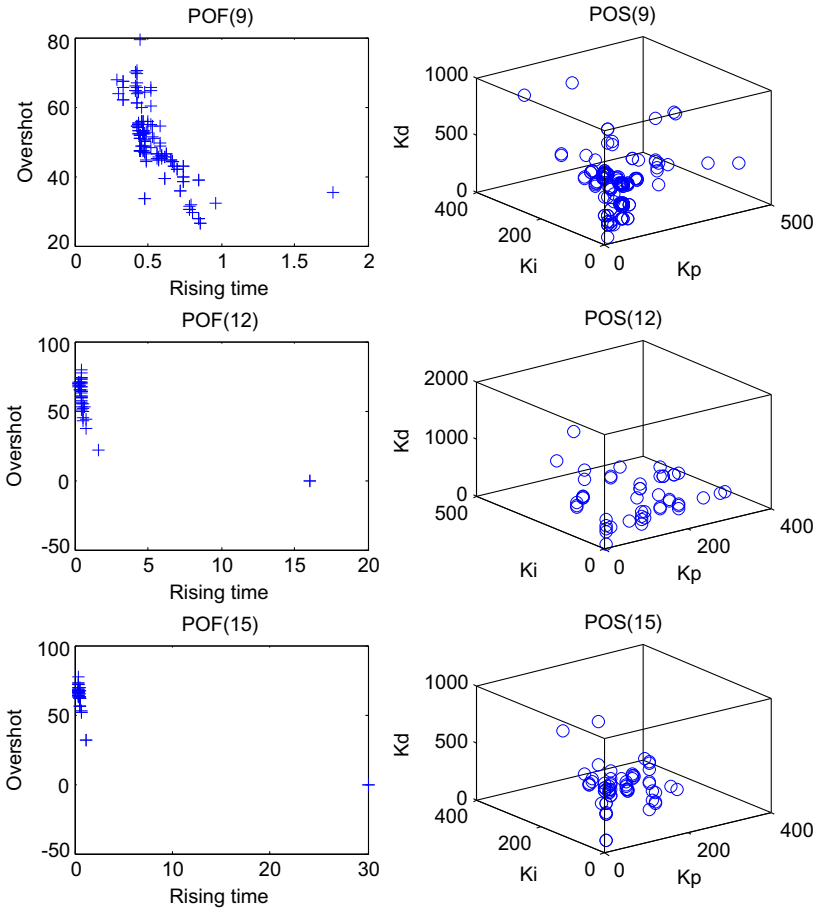

Fig. 9. POS of static problems ($t = 9, 12, 15$ s).

7.1. Stability analysis and search space

The transfer function of the closed-loop system is:

$$H(s) = \frac{G(s)C(s)}{1 + G(s)C(s)} = \frac{1.5(K_d(t)s^2 + K_p(t)s + K_i(t))}{50s^4 + a_2(t)s^3 + (1.5K_d(t) + a_1(t))s^2 + (1.5K_p(t) + 1)s + 1.5K_i(t)}. \tag{19}$$

Its characteristic equation is

$$50s^4 + a_2(t)s^3 + (1.5K_d + a_1(t))s^2 + (1.5K_p + 1)s + 1.5K_i = 0. \tag{20}$$

Since the future model of the plant is unpredictable, the closed-loop control system should have assured stability. For the four-rank characteristic equation, the closed-loop system is stable at any time when the following constraint conditions are satisfied according to the Routh–Hurwitz criterion:

$$a_2(t) > 0; \tag{21}$$
$$1.5K_d(t) + a_1(t) > 0; \tag{22}$$
$$1.5K_p(t) + 1 > 0; \tag{23}$$
$$1.5K_i(t) > 0; \tag{24}$$
$$a_2(t)(1.5K_d(t) + a_1(t)) - 50(1.5K_p(t) + 1) > 0; \tag{25}$$

$$(1.5K_p(t) + 1)(a_2(t)(1.5K_d(t) + a_1(t)) - 50(1.5K_p(t) + 1)) - 1.5K_i a_2^2(t) > 0. \tag{26}$$

The first constraint, (21), is true according to the parameters of the test case. Other constraints, can be rearranged into following formule:

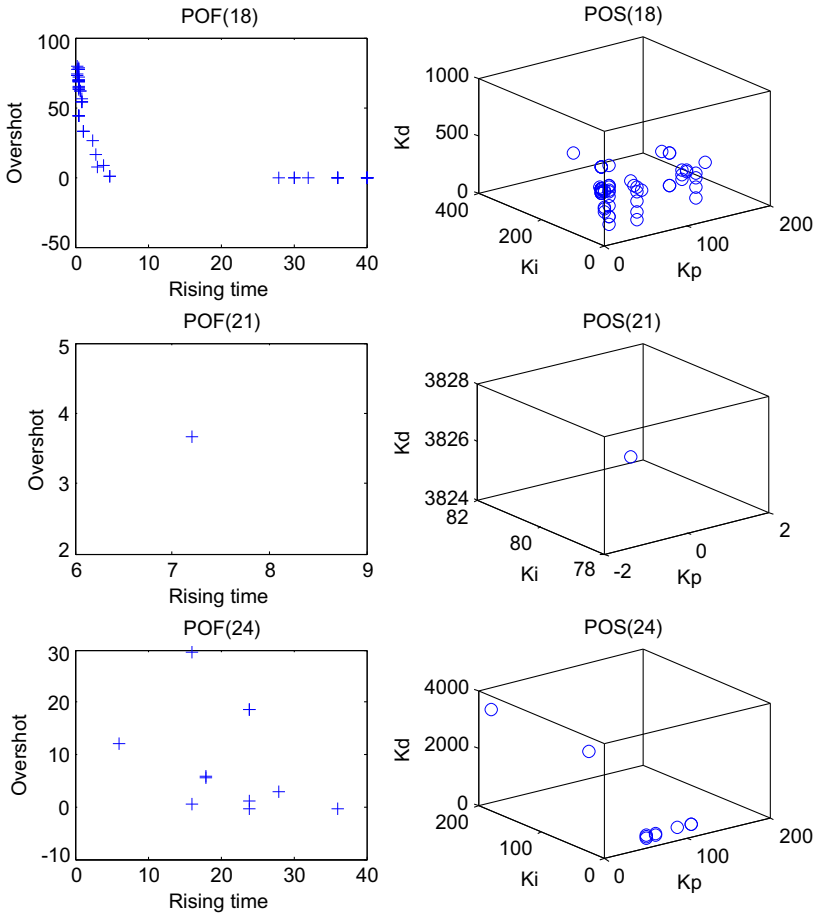

Fig. 10. POS of static problems (t = 18,21,24 s).

$$K_d(t) > -\frac{a_1(t)}{1.5}; \tag{27}$$

$$K_d(t) > \frac{75K_p(t) + 50 - a_1(t)a_2(t)}{1.5a_2(t)}; \tag{28}$$

$$-\frac{1}{1.5} < K_p(t) < \frac{1.5K_d(t)a_2(t) + a_1(t)a_2(t) - 50}{75}; \tag{29}$$

$$0 < K_i(t) < \frac{(1.5K_p(t)+1)(a_2(t)(1.5K_d(t)+a_1(t)) - 50(1.5K_p(t)+1))}{1.5a_2^2(t)}. \tag{30}$$

According to (27), while $f(t) < -\frac{5.3317}{30} \approx -0.17772$, there is some

$$K_d(t) > -a_1(t) = 8.3317. \tag{31}$$

Therefore, If $K_d(t) = 8.3317$ and the plant model keeps stationary, the closed-loop system is unstable. In other words, while $f(t) \in [-1, -0.17772]$, there is no single fixed controller that ensures that the closed-loop system is stable. GA and PSO algorithms failed to find the parameters of a stable controller in this range. Therefore, the fixed parameter $K_d(t) = 8.3317$ [9] is not suitable. Due to a lack of transcendental knowledge about the parameter space, large scale [0,1000] values are searched as illustrated in Figs. 8–11.

When the parameters of a controller are near their critical values, which are constrained by Eqs. (27)–(31), the closed-loop system decays slowly or even oscillates indefinitely. If there are some disturbances on the parameters, the system will become unstable. In addition, although the time-dependent closed-loop system is stable (or unstable) in the current time section, it will be unstable (or stable) in the next time section because the plant is dynamic. For instance, the parameter $a_1(t) = 3 + 30\sin(\pi t/18)$ turns into positive from negative around $t = 35.426$. The plant loses its positive poles and the models for the system are mismatched. Thus, the control performance will further deteriorates further. So as to increase the robustness, the poles of closed system have to be placed into the left plane, far from the imaginary axis. It will be assured that the

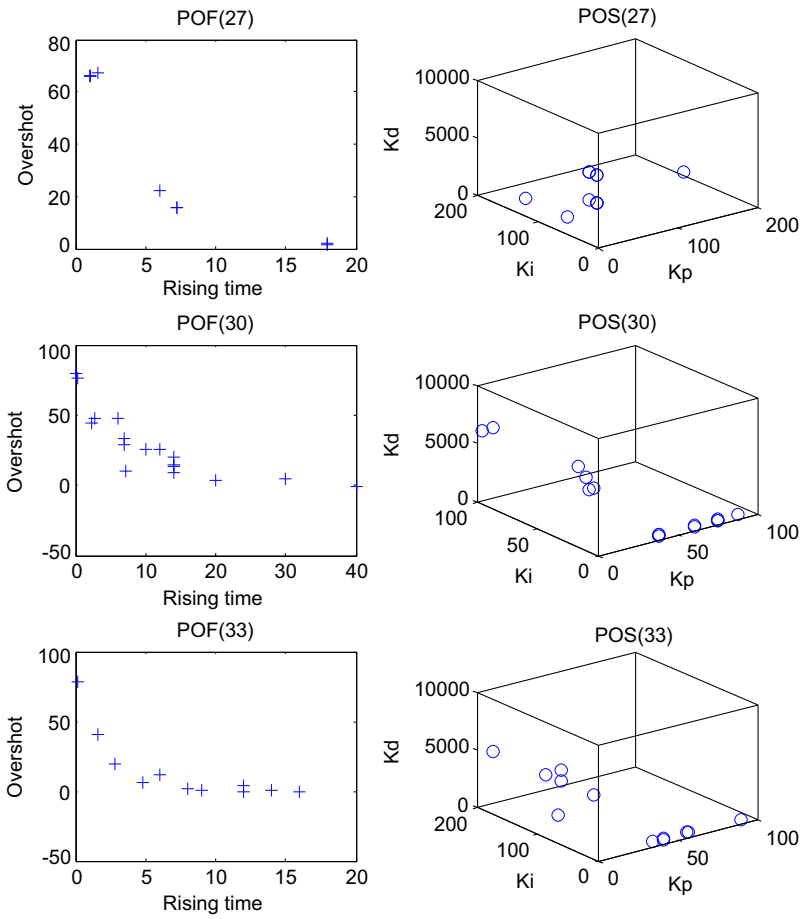

Fig. 11. POS of static problems ($t = 27, 30, 33$ s).

closed-loop system will not be unstable with a small disturbance or change of controller parameter. For example, if the poles are in place left of location $(-1,0)$, then variable s in (20) is replaced by $(z-1)$. The characteristic equation becomes:

$$50z^4 + (a_2(t) - 200)z^3 + (1.5K_d(t) + 300 + a_1(t) - 3a_2(t))z^2 + (1.5K_p(t) - 3K_d(t) - 2a_1(t) - 3a_2(t) + 301)z + 1.5K_i(t) - 1.5K_p(t) + 1.5K_d(t) + a_1(t) - a_2(t) + 48.5 = 0. \quad (32)$$

The necessary condition for a stable system is that the coefficients of the characteristic equations are positive according to the Routh–Hurwitz criterion. It requires that:

$$200 < a_2(t) = 43 + 30f(t) = 43 + 30\sin\left(\frac{\pi t}{18}\right). \quad (33)$$

Obviously, this cannot be true for some values of time. Therefore, the stable margins of the control systems are inadequate for practical engineering. Therefore, the PID controller for the test case is suspect. However, it is a good test case to verify the performance of multi-objective optimization algorithms. This work adopts the PID controller as the basic controller and optimizes its parameters to realize the control of a dynamic unstable plant based on multi-objective optimization algorithms.

7.2. Dynamic superimposition of the static optimal controllers

The result of dynamic multi-objective optimization is not a controller, but a set of non-dominated controller sets as depicted in Figs. 8–11. In the multi-objective optimization controller design process, the trade-off among nondominated controllers is delayed until a set of feasible nondominated controllers is designed. Most investigations are only concerned with the method of obtaining the resultant controller set. The final choice should be decided by the preferences of engineers considering practical matters. In the dynamic case, one controller must be used and only one controller can be adopted at any time. The most difficult challenge is that controllers must be chosen online from these optimal controller sets. There is little knowledge about the forthcoming nondominated controller set. Moreover, there is not enough knowledge of the system's

condition. There is also no sign of the next systemic change. Finally, real-time choice must be made before the next change emerges.

Because nondominated controllers have advantages and disadvantages, the choice of the trade-off controller is difficult. In this control problem, the criteria T_r, δ, and Ts appears in different terms. Therefore, they might be combined as one objective with different priorities at different times.

In order to illustrate the selection and switching of controllers, an experiment is performed as detailed below. In the first step, the controllers with minimum rise time are selected from the non-dominated controller sets when the systemic output is in the rising portion. In the second stage, the controllers with minimum overshoot are chosen before overshoot occurs. In the final stage, the controllers with minimum settling time are adopted from the corresponding static sets. Fig. 12 illustrates the effect of the strategy. However, it is obviously a failure in practice.

To illustrate the reasons for the failure of this strategy, the control of a static plant $G(s) = \frac{1-0.5s}{(s+1)^3}$ is taken as an example. Controllers $C1(s) = 1.8406\left(1 + \frac{1}{1.9523s} + \frac{0.8498s}{0.0269s+1}\right)$ and $C2(s) = 0.4556\left(1 + \frac{1}{1.9292s} + \frac{0.0005s}{4.7159s+1}\right)$ are chosen from the plant's non-dominated solution set. Controller $C1(s)$ has a shorter rising time ($1.9994s$) with overshoot (24.7%); controller $C2(s)$ has a longer rising time ($6.6528s$) without overshoot while they control the plant respectively from zero state. Considered the following strategy, controller $C2(s)$ is switched on in order to decrease the overshoot when the output are adjust from zero to 90% of the given value by controller $C1(s)$. The practical output is shown in Fig. 13. As is evident, there is high overshoot and substantial oscillation. This combination of the two controllers will not improve the performance, but will instead damage it. To the controller $C2(s)$, in fact, the adjust result of $C1(s)$ is similar to a disturbance. In addition, a controller without overshoot has less power to restrain disturbances. As is shown in Fig. 13, it is important to note that the impact on the output is substantial if the controller $C2(s)$ is switched on when the output is unstable (before the adjusting time).

The experiment above illustrates that the optimal controller set of a dynamic system over the entire time axis is not simply the combination of the controllers that come from the static optimal controller sets in each time section. On one hand, the control performance will worsen if the plant changes; new controllers must be adopted for new plants. On the other hand, the dynamic process will quickly worsen if the optimal controller for the zero static plant is switched on during the dynamic process. This is a dilemma.

When a new controller is switched on, the system is not in the zero initial state at which the controller is considered a nondominated one. In other words, the optimal controller set obtained at static state (zero initial state) is not optimal controller to the system in dynamic environment (nonzero initial state). Therefore, the optimal controller set should be recomputed in new dynamic environments. The current state should be taken into consider when optimal controllers are sought online by the dynamic multi-objective optimization algorithm. The optimal controllers based on the current systemic state, historical control rule, and output are very different from the controllers based only on the systemic current model with zero state.

7.3. Dynamic choice of controllers

Considering the plant as static over a small section of time, the methodology [9] searches the optimal set according to three general criteria: R, O, and ST. The series of R, O, and ST in each time section form into the dynamic criteria, $R(t)$, $O(t)$, and $ST(t)$. In practice, the entire systemic output response has only one value each for the three criteria. They are non-dynamic constants: R, O, and ST. Therefore, the evaluation criterion should be adjusted after the common criteria R, O, and ST are met one by one. After the settling time ST, the low oscillation frequency and small error may become the most important

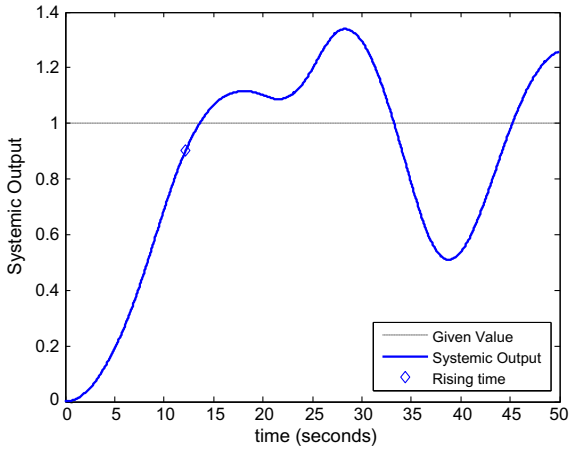

Fig. 12. Superimposition of static controllers.

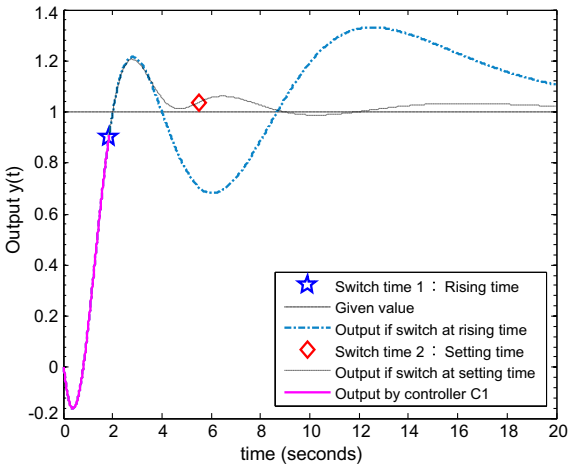

Fig. 13. Effect of controller switching on a static plant.

for good performance. The rising time, overshoot, and settling time based on the current plant model in the zero initial condition can be considered intrinsic and basic requirements. Based on the investigation of the method [15], the three common criteria are modified to adapt to the entire time range. The evaluation functions are modified as follows:

$$\begin{cases} \min.\ \{T_r(t)+J, \delta(t)+J, T_s(t)+J\}; \\ J = 0.01 \times \sum_{t=t_c}^{t_c+t_e} |error(t)|. \end{cases} \tag{34}$$

The objectives $T_r(t)$, δ(t) and $T_s(t)$ are modified as their originals were not defined after the output error is adjusted to less than 2%. The criterion $T_r(t)$ represents the time taken for the output to reach the given value. The criterion $\delta(t)$ is the maximum absolute output error during the given evaluation term t_e after the current time section. The criterion $T_s(t)$ represents the time after which the output error is less than 1%. Each objective adds J as a penalty function. Long or short evaluation time is not applicable. In the following work, the variable is fixed as $t_e = 4$ s. During this time, the plant is considered to be static.

According to the general idea outlined above, DMOAP obtained a series of optimal controller sets at each time section. The control strategy formed over the entire time range by a controller is selected from correspondent nondominated controller set at each time section. Selecting a controller from the resulting nondominated controllers is a difficult problem. Different choices make the next nondominated controller set very different. Moreover, the current nondominated depends on the choice made during the previous time span.

For stationary multi-objective optimization problems, the decision-making is left to the engineers or decision-makers after a nondominated solution set is found. In the dynamic case, while the nondominated solution set is sought, a tradeoff solution must be chosen and implemented for the next optimization phase as soon as possible. Therefore, engineers or decision-makers must be replaced by an automatic procedure to choose a tradeoff solution from the resulting nondominated solutions.

In most situations, the more *a priori* knowledge about the resulting nondominated solution set, the more it helps the building of this type of automatic decision-making mechanism. There are two types of methods in general. The first method is assigning a weight w_i for each objective according to their relative importance. Then, the sum of all objectives is computed. The best solution is selected for the forthcoming time period according to the computed result. For example, the control test problem could choose the controller with lowest J value at each time section as shown in the following:

$$J = w_1 R + w_2 O + w_3 ST. \tag{35}$$

However, some method is required for the confirmation of the dynamic weights. If the weights could be fixed with *a priori* knowledge, the multi-objective optimization becomes a single easy optimization problem.

Another natural method is the one in which a trade-off solution is chosen from the solution set according to additional objectives or secondary objectives. There are many objectives in practical engineering. The least energy required or smallest output error can be introduced as a secondary objective to select a tradeoff controller for the forthcoming period of the control problem.

Different selection principles have important effects on the systemic output performance. Because the previous performance will be optimized later, the systemic output performance is only evaluated in a given time term t_e. The two choice rules are given in Table 3 for different scenarios. As shown in the second row of Table 3, the most important task is calculating the shortest rising time if the output error is larger than 10% of the given value. This rule will choose the controller by which the output approaches the given value during t_e as quickly as possible. If the oscillation of the output appears or the

Table 3
Decision-making table.

Current $error(t)$	r_1: first selection rule (min)	r_2: second selection rule (min)
$\|error(t)\| > 10\%$	$\min(\|error(t)\|),\ t \in [t_c, t_c + t_e]$	$J_T = \int_{t_c}^{t_c+t_e} \mid error(t) \mid dt$
$2\% < \|error(t)\| < 10\%$	$\max(\|error(t)\|),\ t \in [t_c, t_c + t_e]$	$J_T = \int_{t_c}^{t_c+t_e} \mid error(t) \mid dt$
$\|error(t)\| < 2\%$	$J_T = \int_{t_c}^{t_c+t_e} \|error(t)\| dt$	$\max(\|error(t)\|),\ t \in [t_c, t_c + t_e]$

output arrives at the given value, the rule loses its effectiveness because $\min(error(t_r)) = 0$, $t_r \in [t_c, t_c + t_e]$. The secondary rule $r_2 : J_T = \int_{t_c}^{t_c+t_e} |error(t)| dt$ will select the controller for which the output will be stable.

It is necessary to point out that the rules in the decision-making table are different from the criteria in the multi-objectives optimization algorithm. They can be considered as assisting each other. However, the decision-making table is somewhat rough, and it requires further investigation. In particular, this system can be considered as a set of rules in the standard P systems. P systems (membrane computing), as computing devices based on rules, are good at dealing with rules. Therefore, more methods can be borrowed from P systems.

According to the idea outlined above, the DMOAP searches the non-dominated controller set for the system's condition every second. The scale of the chromosome set in DMOAP is fixed at 50. The algorithm outputs the nondominated solution set after five iterations. The systemic output and the controller parameter are shown in Figs. 14 and 15. The rising time, settling time, and overshoot are better than the value of any POF in Figs. 8–11. This illustrates that the control and optimization strategies are effective.

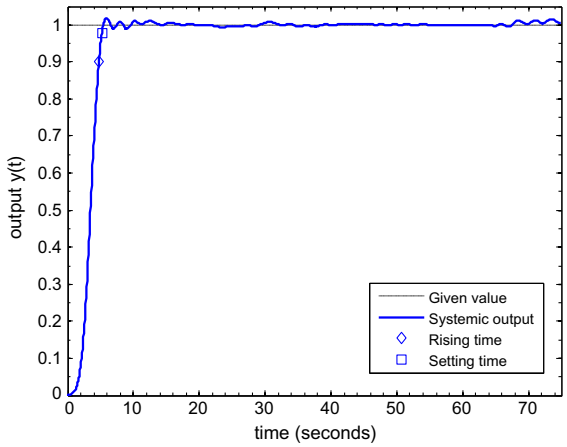

Fig. 14. Optimal control performance.

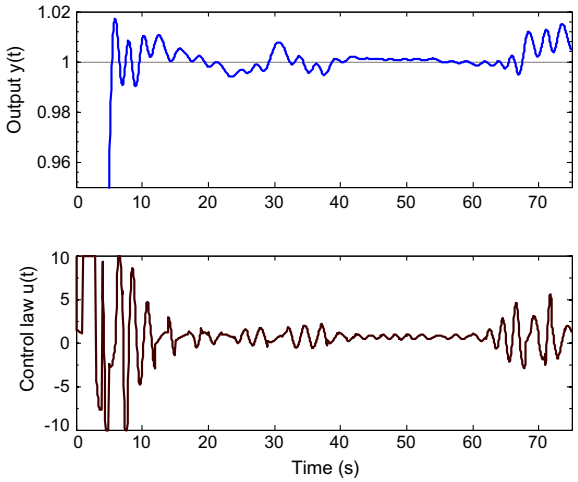

Fig. 15. Output and control law.

7.4. Real time

The unstable dynamic plant is correctly controlled by the control strategy above. However, reliability and real-time performance should be discussed. The control strategy above is realized by the controller parameters are optimized based on the system's condition. The system's condition is adjusted by the new optimized controller. Probably, this is impractical because the optimization algorithm takes time to run. The control strategy requires that controller be optimized in advance. Thus, the plant model and systemic state should be predicted in advance. However, process time and predictive error maybe cause difficult. Therefore, the real time should be checked in particular.

A strategy is that the old controller is retained until a new controller is computed. During the optimization process, the plant model and systemic state change. Therefore, it should be tested whether the new systemic state can be controlled correctly by the new controller. Additionally, the overshoot performance should be experimentally verified.

A new controller is switched on after different delay times (considered to be the optimization times). The object set size of the DMOAP is 100. It iterates six times to produce a new controller. The systemic output performance is shown in Fig. 17, in which a new controller is switched in after different given times. It shows that performance is adequate if a new controller can be optimized in 1.5 s. If the optimization time is longer than 2 s, output performance will worsen. In practice, the simulation optimization time is 0.48 s by Matlab7.1 software on a personal computer with a CPU 2.4 GHz and with 512 M of RAM. The optimization time is less than the limiting 1.5 s. Moreover, the optimization algorithm can be realized in parallel by multiple of computers. If each membrane is simulated by a separate CPU and the procedure is written in C, the control strategy will satisfy real-time requirements [38].

7.5. Reliability problem

Although the output performance is very good and the real-time requirement is met, the reliability bears some discussion. First, the control strategy depends on the dynamic multi-objective optimization algorithm. Although the DMOAP converges quickly and precisely, it is a random, and nondetermined search algorithm. DMOAP fails to assure that it will find a satisfactory controller each time. However, an unsatisfactory controller will be replaced at the next switch cycle. Second, the systemic stability should be investigated while the controller make each switch.

Therefore, although the plant changes continuously, the switch number should be decreased as much as possible. When dynamic multi-objective optimization methods are applied in the controller design above, the control problem is considered either a type I or type II dynamic multi-objective optimization problem in which the non-dominated controller (POS) has changed with the varying plant. Therefore, the control strategy is searching for the dynamic optimal solution set $POS(t)$. The $POF(t)$ is obtained by taking the optimal solution at different times as the parameters of the controller (PID).

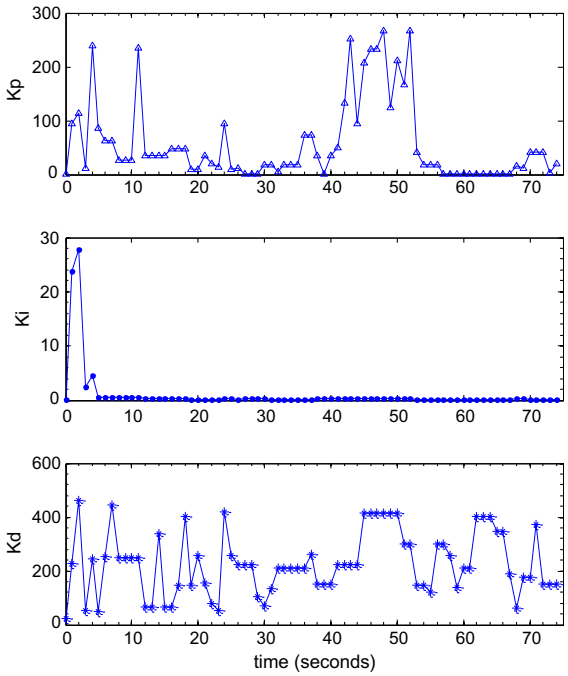

Fig. 16. PID parameters.

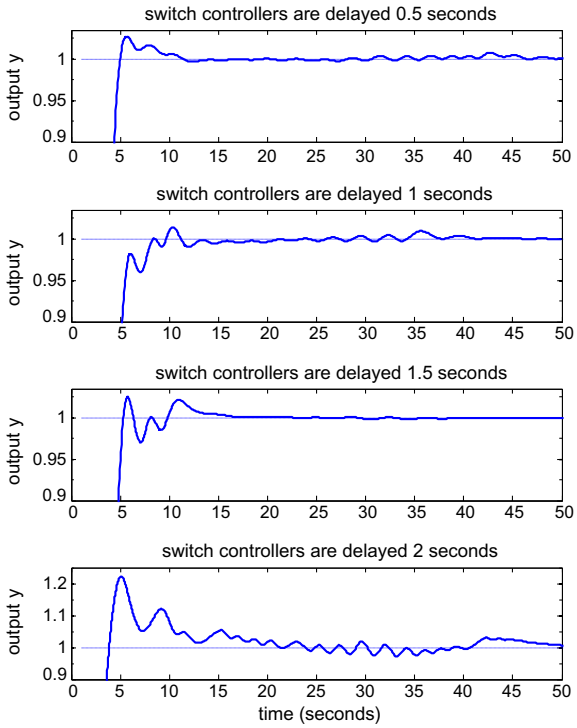

Fig. 17. Systemic output with different optimal times.

In fact, the control problem also can be considered as type IV multi-objective optimization problem in which the POS does not change despite the optimization problem changing. The optimal controller set is computed, which will not change while the plant is changing. The optimal performance in the entire time range is emphasized. In fact, the strategy becomes a robust control strategy to a certain extent. The output is required to satisfy performance without tuning the controller when disturbance occurs.

Since this type of POS will not change, the POS can be sought by the multi-objective optimization algorithm if the change of in the plant can be predicted or simulated. The periodic plant model is easily recognized online. If the dynamic plant is nonperiodic, a longer term is given. Where the recognition time is longer, the model will be more precise. The controllers (solutions) are evaluated based on the recognized dynamic model. The model is verified when controllers are optimized based on recognized models. If the plant changes dramatically, the plant model will be rebuilt for the entire time range and the optimization algorithm will restart. The obtained nondominated solutions (controllers) are taken as initial candidate solutions. At the decision-making stage, the controllers have higher priority, which appears more times as nondominated solutions. The controllers may be made more robust. This robust version is called as the DPMOA-III/IV, which searches a constant POS for the dynamic plant during the entire range. Correspondingly, a controller that searches a temporarily $POS(t)$ during each small time span is called a DPMOA-I/II.

After the system completes a cycle, the dynamic plant is recognized. Based on the dynamic plant, the DPMOA-III/IV gains a set of nondominated robust controllers (POS) if there are stable controllers. For this type of unstable dynamic plant, the systemic performance is usually not good enough despite finding a set of stable controllers. The rising time and settling time may be long and the overshoot may be large.

The dynamic plant can be adjusted by these controllers together, which are optimized by the DPMOA-III/IV and DPMOA-I/II. If the model mismatch is not serious or disturbance is slight, the controller optimized by DPMOA-III/IV is switched on while the system is stable. Thus, the controller switch number will be decreased, and reliability will be increased. If the model mismatch is serious and the current controller fails to restrain it, a series of controllers are switched on, which are optimized by the DPMOA-I/II.

As shown in Fig. 18, the control strategy has two multi-objective optimization sub-systems. The first subsystem optimizes the controller parameters based on the system's condition for temporary control. It takes short time intervals but has strict requirements for the hardware. Moreover, the control strategy depends on system's precise states and switches controllers frequently. The closed-loop system has perfect dynamic output with short rising time and small overshoot. The secondary subsystem searches for a robust controller for long term control. However, it may fail to find a robust controller. Moreover, the secondary subsystem optimizes slowly but does not depend on the systemic current state. The closed-loop system is not sensitive to the change of plant model. The controller is suitable to the stable scenario. The two optimization subsystems

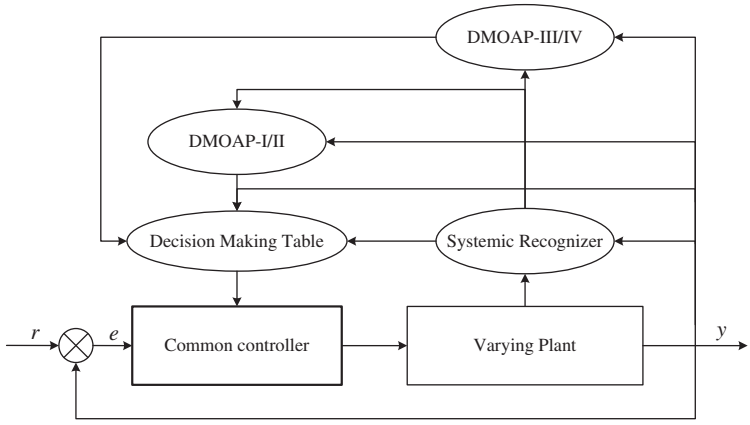

Fig. 18. Membrane control strategy.

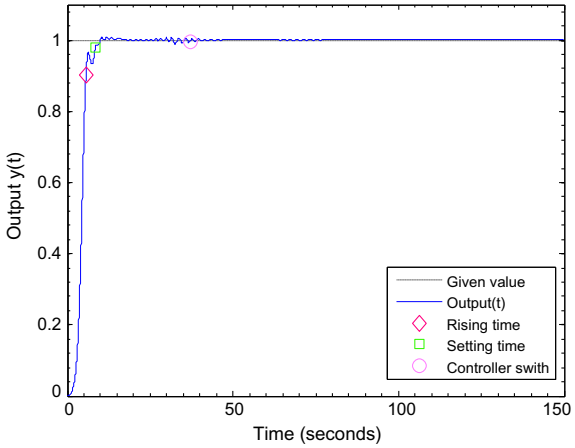

Fig. 19. Control performance (a).

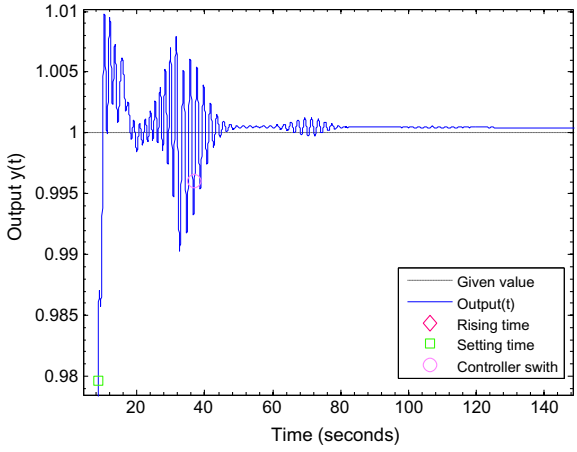

Fig. 20. Control performance (b).

adjust the parameters of a common controller. Therefore, the decision-making is slightly more complicated than depicted in Table 3. Usually, the controller optimized by the first system adjusts the dynamic process. The controller optimized by the secondary subsystem adjusts the stable process. The integrated control strategy has high-quality performance that is

Table 4
Varying controller parameters.

Switch time t_c(s)	K_p	K_i	K_d
0	849.7982	510.9858	129.6465
1	1.060991	2.662683	72.63262
3	101.8757	12.7133	197.4475
5	296.2907	4.763173	417.0785
7	187.3915	1.063664	395.141
9	261.6264	0.075171	565.4168
11	98.82025	0.075171	565.4168
13	95.67501	0.31902	407.8749
15	4.837464	0.285681	996.0672
17	27.90108	0.285681	769.9372
19	201.6736	0.259656	906.9537
21	84.81614	0.259656	539.0685
23	1.298204	0.259656	539.0685
25	4.521637	0.083473	372.2638
27	2.083353	0.152096	372.2638
29	0.155093	0.152096	372.2638
31	0.155093	0.152096	624.6839
33	0.155093	0.152096	624.6839
35	0.155093	0.152096	274.8282
After 36 s	108.95	0.165	340.37

illustrated by performing simulation experiments, as shown in Figs. 19 and 20. The concrete parameters of the controller and switch time are shown in Table 4. Since the algorithm is inspired by membrane computing, the control strategy is called the 'membrane strategy'.

In short, according to the analysis of the stability, the parameter range should be enlarged. With systemic changes in parameter $f(t)$, the series of static POFs will be worse and might not even exist in the domain under consideration. As shown in Fig. 16, if the parameters were constrained to small domains ($K_d(t) = 8.3317$, $K_i(t) \in (0.5, 5.0)$, $K_p(t) \in (0.1, 1.0)$) without transcendental knowledge, the global optimal solutions might be excluded from the given domain. In fact, algorithms GA and PSO failed to find an effective controller that might have been able to realize stable control of the dynamic unstable plant. Based on this investigation, a set of concrete control parameters is designed and shown in Table 4. This control parameters give excellent control performance.

8. Conclusions

This paper investigated several issues related to the dynamic multi-objective optimization problem. We discussed several test cases and also proposed a novel multi-objective optimization algorithm. Moreover, the paper investigated the effects of trade-off solutions for dynamic optimization results. The key process of the dynamic multi-objective optimization is about the construction of discrete, equivalent, and static subproblems during each time range. Issues related to the false solutions, the loss of nondominated solutions, and the dependence on trade-off solution require more future investigation.

Finally, the application of multi-objective optimization in control science is analyzed by taking the varying plant as a test case. The most important contribution is a membrane control strategy for the unstable varying plant that has been designed on the basic of dynamic multi-objective optimization. The stability, real-time performance, controller switching, selection of solution, and reliability were investigated. Simulation results illustrate that the proposed strategy exhibits excellent performance, including ideal systemic output, whereas a GA and PSO algorithm failed to design a stable controller for the unstable time-varying plant.

Acknowledgement

This work was supported in part by the Technology Innovation Program of MKE/KEIT [2008-F-038-01, Development of Context Adaptive Cognition Technology].

References

[1] M.A. Abo-Sinna, Multiple objective (fuzzy) dynamic programming problems: a survey and some applications, Applied Mathematics and Computation 157 (3) (2004) 861–888.
[2] Z. Bingul, Adaptive genetic algorithms applied to dynamic multiobjective problems, Applied Soft Computing 7 (3) (2007) 791–799.
[3] E. Bolinger, An optimization of the fuzzy control algorithm, Information Sciences – Applications 2 (3) (1994) 135–142.
[4] M. Cardona, M.A. Colomer, M.J. Pérez-Jiménez, A. Zaragoza, Handling Markov chains with membrane computing, in: Unconventional Computation, Proceedings, vol. 4135, 2006, pp. 72–85.
[5] O. Castillo, R. Martínez-Marroquín, P. Melin, F. Valdez, J. Soria, Comparative study of bio-inspired algorithms applied to the optimization of type-1 and type-2 fuzzy controllers for an autonomous mobile robot, Information Sciences, in press, Corrected Proof.

[6] W.D. Chang, S.P. Shih, PID controller design of nonlinear systems using an improved particle swarm optimization approach, Communications in Nonlinear Science and Numerical Simulation 15 (11) (2010) 3632–3639.
[7] K. Deb, Multi-Objective Optimization Using Evolutionary Algorithms, Wiley, Chichester, UK, 2001.
[8] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197.
[9] M. Farina, K. Deb, P. Amato, Dynamic multiobjective optimization problems: test cases, approximations, and applications, IEEE Transactions on Evolutionary Computation 8 (5) (2004) 425–442.
[10] E. Fernandez, E. Lopez, F. Lopez, C.A.C. Coello, Increasing selective pressure towards the best compromise in evolutionary multiobjective optimization: the extended NOSGA method, Information Sciences 181 (1) (2011) 44–56.
[11] L. Gosselin, M. Tye-Gingras, F. Mathieu-Potvin, Review of utilization of genetic algorithms in heat transfer problems, International Journal of Heat and Mass Transfer 52 (9-10) (2009) 2169–2188.
[12] S.U. Guan, Q. Chen, W.T. Mo, Evolving dynamic multi-objective optimization problems with objective replacement, Artificial Intelligence Review 23 (3) (2005) 267–293.
[13] H.W. Hayley, T. Jules, Multi-objective optimization for chemical processes and controller design: approximating and classifying the Pareto domain, Computers & Chemical Engineering 30 (6–7) (2006) 1155–1168.
[14] F. Hoffmann, G. Pfister, Evolutionary design of a fuzzy knowledge base for a mobile robot, International Journal of Approximate Reasoning 17 (4) (1997) 447–469.
[15] L. Huang, N. Wang, An optimization algorithm inspired by membrane computing, Advances in Natural Computation 4222 (2006) 49–52.
[16] L. Huang, N. Wang, J.H. Zhao, Multiobjective optimization for controller design, Acta Automatica Sinica 34 (4) (2008) 472–477.
[17] Y.C. Jin, B. Sendhoff, Constructing dynamic optimization test problems using the multi-objective optimization concept, Lecture Notes in Computer Science 3005 (2004) 525–536.
[18] I. Kitsios, T. Pimenides, H[infinity] controller design for a distillation column using genetic algorithms, Mathematics and Computers in Simulation 60 (3) (2002) 357–367.
[19] C.H. Lee, Low order robust controller design for preserving H[infinity] performance: genetic algorithm approach, ISA Transactions 43 (4) (2004) 539–547.
[20] D. Li, Y.Y. Haimes, Multiobjective dynamic programming: the state of the art, Theory and Advanced Technology 5 (4) (1989) 471–483.
[21] C. Martin-Vide, A. Paun, G. Paun, On the power of P systems with symport rules, Journal of Universal Computer Science 8 (2) (2002) 317–331.
[22] M. Mucientes, D.L. Moreno, A. Bugarín, S. Barro, Design of a fuzzy controller in mobile robotics using genetic algorithms, Applied Soft Computing 7 (2) (2007) 540–546.
[23] I. Mukherjee, P.K. Ray, A review of optimization techniques in metal cutting processes, Computers & Industrial Engineering 50 (1-2) (2006) 15–34.
[24] T.Y. Nishida, An application of P system: a new algorithm for NP-complete optimization problems, in: Proceedings of the 8th World Multi-Conference on Systems, Cybernetics and Informationtics, vol. 5, 2004, pp. 109–112.
[25] G. Paun, Computing with membranes, Journal of Computer and System Sciences 61 (1) (2000) 108–143.
[26] G. Paun, From cells to computers: computing with membranes (P systems), Biosystems 59 (3) (2001) 139–158.
[27] G. Paun, Membrane computing: power, efficiency, applications, New Computational Paradigms 3526 (2005) 396–407.
[28] G. Paun, M.J. Pérez-Jiménez, Membrane computing: brief introduction, recent results and applications, Biosystems 85 (1) (2006) 11–22.
[29] A. Paun, G. Paun, The power of communication: P systems with symport/antiport, New Generation Computing 20 (3) (2002) 295–305.
[30] N. Serinivas, K. Deb, Multiobjective optimization using nondominated sorting in genetic algorithms, Evolutionary Computation 2 (3) (1994) 221–248.
[31] C. Shi, Z.Y. Yan, K. Lü, Z.Z. Shi, B. Wang, A dominance tree and its application in evolutionary multi-objective optimization, Information Sciences 179 (20) (2009) 3540–3560.
[32] V.V.R. Silva, P.J. Fleming, J. Sugimoto, R. Yokoyama, Multiobjective optimization using variable complexity modelling for control system design, Applied Soft Computing 8 (1) (2008) 392–401.
[33] R.R. Sumar, A.A.R. Coelho, L.D.S. Coelho, Use of an artificial immune network optimization approach to tune the parameters of a discrete variable structure controller, Expert Systems with Applications 36 (3) (2009) 5009–5015.
[34] F. Szidarovszky, L. Duckstein, Dynamic multiobjective optimization: a framework with application to regional water and mining management, European Journal of Operational Research 24 (2) (1986) 305–317.
[35] A. Tarafder, G.P. Rangaiah, A.K. Ray, A study of finding many desirable solutions in multiobjective optimization of chemical processes, Computers & Chemical Engineering 31 (10) (2007) 1257–1271.
[36] K. Trojanowski, S.T. Wierzchon, Immune-based algorithms for dynamic optimization, Information Sciences 179 (10) (2009) 1495–1515.
[37] V.M. Zavala, A. Flores-Tlacuahuac, E. Vivaldo-Lima, Dynamic optimization of a semi-batch reactor for polyurethane production, Chemical Engineering Science 60 (11) (2005) 3061–3079.
[38] E. Zitzler, K. Deb, L. Thiele, Comparison of multi-objective evolutionary algorithms: empirical results, IEEE Transactions on Evolutionary Computation 8 (2) (2000) 173–195.
[39] F.A. Zotes, M.S. Peñas, Multi-criteria genetic optimisation of the manoeuvres of a two-stage launcher, Information Sciences 180 (6) (2010) 896–910.