Continuous Dynamic Constrained Optimization—The Challenges

Trung Thanh Nguyen, *Member, IEEE,* and Xin Yao, *Fellow, IEEE*

***Abstract*—Many real-world dynamic problems have constraints, and in certain cases not only the objective function changes over time, but also the constraints. However, there is no research in answering the question of whether current algorithms work well on continuous dynamic constrained optimization problems (DCOPs), nor is there any benchmark problem that reflects the common characteristics of continuous DCOPs. This paper contributes to the task of closing this gap. We will present some investigations on the characteristics that might make DCOPs difficult to solve by some existing dynamic optimization (DO) and constraint handling (CH) algorithms. We will then introduce a set of benchmark problems with these characteristics and test several representative DO and CH strategies on these problems. The results confirm that DCOPs do have special characteristics that can significantly affect algorithm performance. The results also reveal some interesting observations where the presence or combination of different types of dynamics and constraints can make the problems easier to solve for certain types of algorithms. Based on the analyses of the results, a list of potential requirements that an algorithm should meet to solve DCOPs effectively will be proposed.**

***Index Terms*—Benchmark problems, constraint handling (CH), dynamic constraints, dynamic environments, dynamic optimization (DO), evolutionary algorithms, performance measures.**

I. Introduction

THIS PAPER aims to answer some open questions about the characteristics, difficulties and solutions of a very common class of problems—dynamic constrained optimization problems (DCOPs). DCOPs are constrained optimization problems that have two properties: 1) the objective functions, the constraints, or both, may change over time; and 2) the changes are taken into account in the optimization process.[1] It is believed that a majority of real-world dynamic problems are DCOPs. However, there are few studies on continuous dynamic constrained optimization. Existing studies on continuous dynamic optimization only focus on the unconstrained or domain constraint dynamic cases (which in this paper both are regarded as "unconstrained" problems). Likewise, existing research in constraint handling only focuses on the stationary constrained problems.

This lack of attention on DCOPs in the continuous domain raises some important research questions: What are the essential characteristics of these types of problems? How well would existing dynamic optimization and constraint handling strategies perform in dynamic constrained environments if most of them are designed for and tested in either unconstrained dynamic problems or stationary constrained problems only? Why do they work well or not? How can one evaluate if an algorithm works well or not? And finally, what are the requirements for a "good" algorithm that effectively solves these types of problems?

As a large number of real-world applications are dynamic constrained, finding the answers to the above questions is essential. Such answers would help to have better understanding about the practical issues of DCOPs and to solve this class of problems more effectively.

This paper is organized as follows. Section II identifies the special characteristics from real-world DCOPs and discusses how the characteristics make this type of problem different from unconstrained dynamic optimization problems (DOPs). Section III reviews related literature about continuous benchmark problems, identifies the gaps between them and real-world problems, and proposes a new set of DCO benchmark problems. Sections IV and V investigate the possibility of solving DCOPs using some representative DO/CH strategies. Experimental analyses about the strengths and weaknesses, and the effect of the mentioned characteristics on each strategy will be undertaken. Based on the experimental results, a list of requirements that algorithms should meet to solve DCOPs effectively are proposed. Finally, Section VI concludes this paper and points out future directions of research.

II. Characteristics of Real-World Dynamic Constrained Problems

The presence of constraints in DCOPs from real-world applications makes them very different from the unconstrained or domain constraint problems considered in academic research. In real-world DCOPs, the objective function and constraint functions can be combined in three different types: 1) both

Manuscript received November 14, 2011; accepted November 16, 2011. Date of publication February 7, 2012; date of current version November 27, 2012. This work was supported by an U.K. ORS Award, a studentship from the School of Computer Science, University of Birmingham, and two EPSRC Grants EP/E058884/1 and EP/D052785/1.

T. T. Nguyen is with the School of Engineering, Technology and Maritime Operations, Liverpool John Moores University, Liverpool L3 3AF, U.K. (e-mail: t.t.nguyen@ljmu.ac.uk).

X. Yao is with the Center of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K. (e-mail: x.yao@cs.bham.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2011.2180533

[1]This definition is derived from the (more general) definition of dynamic optimization problems in [1, Sec. V].

the objective function and the constraints are dynamic [2]–[4]; 2) only the objective function is dynamic while the constraints are static [5]–[7]; and 3) the objective function is static and the constraints are dynamic [8]–[10]. In all three types, the presence of infeasible areas can affect how the global optimum moves, or appears after each change. This leads to some special characteristics which are not found in the unconstrained and fixed constrained cases.

First, constraint dynamics can lead to changes in the shape/percentage/structure of the feasible/infeasible areas. Second, objective function dynamics might cause the global optima to switch from one disconnected feasible region to another on problems with disconnected feasible regions, which are very common in real-world constrained problems, especially the scheduling problems [11]–[13]. Third, in problems with fixed objective functions and dynamic constraints, the changing infeasible areas might expose new, better global optima without changing the existing optima. One example is the dynamic 0–1 Knapsack problem: significantly increasing the capacity of the knapsack can create a new global optimum without changing the existing optimum.

In addition to the three special characteristics above, DCOPs might also have the common characteristics of constrained problems such as global optima in the boundaries of feasible regions, global optima in search boundary, and multiple disconnected feasible regions. These characteristics are widely regarded as being common in real-world applications.

III. Real-Valued Benchmark to Simulate DCOPs' Characteristics

A. Related Literature

In the continuous domain, there is no existing continuous benchmark that fully reflects the characteristics of DCOPs listed in Section II. Among existing continuous benchmarks, there are only two recent studies that are related to dynamic constraints. The first study was [14], in which two simple unimodal constrained problems were proposed. These problems take the time variable t as their only time-dependant parameter and hence the dynamic was created by the increase over time of t. These problems have some important disadvantages which prevent them from being used to capture/simulate the mentioned properties of DCOPs: they only capture a simple linear change. In addition, the two problems do not reflect common situations like dynamic objective + fixed constraints or fixed objective + dynamic constraints and other common properties of DCOPs.

The second study was [15]. In that research, a dynamic constrained benchmark problem was proposed by combining an existing "field of cones on a zero plane" dynamic fitness function with four dynamic norm-based constraints with the square/diamond/sphere-like shapes (see [15, Fig. 2]). Although the framework used to generate this benchmark problem is highly configurable, the current single benchmark problem generated by the framework in [15] was designed for a different purpose and hence does not simulate the properties mentioned in Section II. For example, the benchmark problem might not be able to simulate common properties of DCOPs such as optima in boundary, disconnected feasible regions, and moving constraints exposing optima in a controllable way. In addition, there is only one single type of benchmark problem and hence, it might be difficult to use the problem to evaluate the performance of algorithms under different situations.

The lack of benchmark problems for DCOPs makes it difficult to: 1) evaluate how well existing DO algorithms would work on DCOPs; and 2) design new algorithms specializing in DCOPs. Given that a majority of recent real-world DOPs are DCOPs [16], this can be considered an important gap in DO research.

This gap motivates the authors to develop general-purpose benchmark problems to capture the special characteristics of DCOPs. Some initial results involving five benchmark problems were reported in an earlier study [17]. This paper extends the framework to develop full sets of benchmark problems, which are able to capture all characteristics mentioned in the previous section. Two sets of benchmark problems, one with multimodal, scalable objective functions and one with unimodal objective functions, have been developed for this research. In this paper, the benchmark set with unimodal objective functions (many problems in the set still have multiple optima due to the constraints) will be discussed in detail. Detailed descriptions of the multimodal, scalable set can be found in a technical report [18].

B. Generating Dynamic Constrained Benchmark Problems

One useful way to create dynamic benchmark problems is to combine existing static benchmark problems with the dynamic rules found in dynamic constrained applications. This can be done by applying the dynamic rules to the parameters of the static problems, as described below.

Given a static function $f_P(x)$ with a set of parameters $P = \{p_1, ..., p_k\}$, one can always generalize $f_P(x)$ to its dynamic version $f_{P_t}(x, t)$ by replacing each static parameter $p_i \in P$ with a time-dependent expression $p_i(t)$. The dynamic of the dynamic problem then depends on how $p_i(t)$ varies over time. One can use any type of dynamic rule to represent $p_i(t)$, and hence can create any type of dynamic problem. Details of the concept and a mathematical framework for the idea is described in [18]. Some additional information is provided in [19, Sec. 3].

C. Dynamic Constrained Benchmark Set

A set of 18 benchmark problems named G24 was introduced using the new procedure described in the previous subsection. The general form for each problem in the G24 set is as follows:

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & g_i(\mathbf{x}) \le 0 \quad g_i(\mathbf{x}) \in G \quad i = 1, ..., n \end{array}$$

where the objective function $f(\mathbf{x})$ can be one of the function forms set out in (1), each constraint $g_i(\mathbf{x})$ can be one of the function forms given in (2), and G is the set of n constraint functions for that particular benchmark problem. The detailed descriptions of $f(\mathbf{x})$ and $g_i(\mathbf{x})$ for each problem are described in Tables I and II.

Equation (1) describes the general function forms for the objective functions in the G24 set. Of these function forms,

TABLE I
OBJECTIVE FUNCTION FORM OF EACH BENCHMARK PROBLEM

Benchmark Problem	Objective Function
G24_8a and G24_8b	$f(x) = f^{(2)}$
All other problems	$f(x) = f^{(1)}$

TABLE II
SET OF CONSTRAINT FUNCTION FORMS FOR EACH PROBLEM

Benchmark Problem	Set G of Constraints
G24_u, G24_uf, G24_2u, G24_8a	$G = \{\emptyset\}$
G24_6a	$G = \{g^{(3)}, g^{(6)}\}$
G24_6b	$G = \{g^{(3)}\}$
G24_6c	$G = \{g^{(3)}, g^{(4)}\}$
G24_6d	$G = \{g^{(5)}, g^{(6)}\}$
All other problems	$G = \{g^{(1)}, g^{(2)}\}$

$f^{(2)}$ is used to design the objective function for G24_8a and G24_8b, and $f^{(1)}$ is used to design the objective functions for all other problems. $f^{(1)}$ is modified from a static function proposed in [20] and $f^{(2)}$is a newly designed function

$$\begin{aligned} f^{(1)} &= -(X_1 + X_2) \quad (1)\\ f^{(2)} &= -3\exp\left(-\sqrt{\sqrt{(X_1)^2 + (X_2)^2}}\right) \end{aligned}$$

where $X_i = X_i(x_i, t) = p_i(t)(x_i + q_i(t))$, $0 \le x_1 \le 3, 0 \le x_2 \le 4$ with $p_i(t)$ and $q_i(t)$ $(i = 1, 2)$ as the dynamic parameters, which determine how the dynamic objective function of each benchmark problem changes over time.

Equation (2) describes the general function forms for the constraint functions in the G24 set. Of these function forms, $g^{(1)}$and $g^{(2)}$were modified from two static functions proposed in [20] and $g^{(3)}$, $g^{(4)}$, $g^{(5)}$and $g^{(6)}$are newly designed functions

$$\begin{aligned} g^{(1)} &= -2Y_1^4 + 8Y_1^3 - 8Y_1^2 + Y_2 - 2 \quad (2)\\ g^{(2)} &= -4Y_1^4 + 32Y_1^3 - 88Y_1^2 + 96Y_1 + Y_2 - 36\\ g^{(3)} &= 2Y_1 + 3Y_2 - 9\\ g^{(4)} &= \begin{cases} -1, & \text{if } (0 \le Y_1 \le 1) \text{ or } (2 \le Y_1 \le 3)\\ 1, & \text{otherwise} \end{cases}\\ g^{(5)} &= \begin{cases} -1, & \text{if } (0 \le Y_1 \le 0.5) \text{ or } (2 \le Y_1 \le 2.5)\\ 1, & \text{otherwise} \end{cases}\\ g^{(6)} &= \begin{cases} -1, & \text{if } [(0 \le Y_1 \le 1) \text{ and } (2 \le Y_2 \le 3)]\\ & \text{or } (2 \le Y_1 \le 3)\\ 1, & \text{otherwise} \end{cases} \end{aligned}$$

where $Y_i = Y_i(x, t) = r_i(t)(x + s_i(t))$, $0 \le x_1 \le 3, 0 \le x_2 \le 4$ with $r_i(t)$, and $s_i(t)$ $(i = 1, 2)$ as the dynamic parameters, which determine how the constraint functions of each benchmark problem change over time.

Each benchmark problem may have a different mathematical expression for $p_i(t)$, $q_i(t)$, $r_i(t)$ and $s_i(t)$. Note that although many benchmark problems share the same general function form in (1), their individual expressions for $p_i(t)$ and $q_i(t)$ make their actual dynamic objective functions very different. Similarly, the individual expressions for $r_i(t)$ and $s_i(t)$ make each actual dynamic constraint function very different although they may share the same function form. The individual expressions of $p_i(t)$, $q_i(t)$, $r_i(t)$, and $s_i(t)$ for each benchmark function are described in Table III.

Two guidelines were used to design the test problems: 1) problems should simulate the common properties of DCOPs as mentioned in Section II; and 2) there should always be a pair of problems for each characteristic. The two problems in each pair should be almost identical except that one has a particular characteristic (e.g., fixed constraints) and the other does not. By comparing the performance of an algorithm on one problem with its performance on the other problem in the pair, it is possible to analyze whether the considered characteristic has any effect on the tested algorithm and to what extent that effect is significant.

Based on the two guidelines above, 18 different test problems were created (Table III). Each test problem is able to capture one or several of the mentioned characteristics of DCOPs, as shown in Table IV. In addition, the problems and their relationships are carefully designed so that they can be arranged in 21 pairs (Table V), of which each pair is a different test case to test a single characteristic of DCOPs (the two problems in each pair are almost identical except that one has a special characteristic and the other does not).

IV. CHALLENGES OF APPLYING CURRENT DYNAMIC OPTIMIZATION STRATEGIES DIRECTLY TO SOLVING DCOPS

A. *Analyzing the Performance of Some Common Dynamic Optimization Strategies in Solving DCOPs*

The strategies being considered are: 1) introducing diversity; 2) maintaining diversity; and 3) tracking the previous optima. These three are among the four most commonly used strategies (the other strategy is memory-based) to solve DOPs. The diversity-introducing strategy was proposed based on the assumption that by the time a change occurs in the environment, an evolutionary algorithm might have already converged to a specific area and hence would lose its ability to deal with changes in other areas of the search space. Consequently, it is necessary to increase the diversity level in the population, either by increasing the mutation rate or re-initializing/re-locating the individuals. This strategy was introduced years ago [21] but is still extensively used [22], [23].

The diversity-introducing strategy requires that changes must be visible to the algorithm. To avoid this disadvantage, the diversity-maintaining strategy was introduced so that population diversity can be maintained without explicitly detecting changes [24]. This strategy is still the main strategy in many recent approaches [25], [26].

The third strategy, tracking-previous-optima, is used where the optima might only slightly change. The region surrounding the current optima is monitored to detect changes and "track" the movement of these optima. Similar to the two strategies above, the tracking strategy has also been used for years [21] and it has always been one of the main strategies for solving DOPs. Recently, this strategy has been combined with the diversity maintaining/introducing strategy

TABLE III
DYNAMIC PARAMETERS FOR ALL TEST PROBLEMS IN THE BENCHMARK SET G24

Prob	Parameter Settings
G24_u	$p_1(t) = \sin\left(k\pi t + \frac{\pi}{2}\right),\ p_2(t) = 1,\ q_i(t) = 0$
G24_1	$p_2(t) = r_i(t) = 1,\ q_i(t) = s_i(t) = 0$ $p_1(t) = \sin\left(k\pi t + \frac{\pi}{2}\right)$
G24_f	$p_i(t) = r_i(t) = 1,\ q_i(t) = s_i(t) = 0$
G24_uf	$p_i(t) = 1,\ q_i(t) = 1$

G24_2	if $(t \bmod 2 = 0)$ $\begin{cases} p_1(t)=\sin\left(\frac{k\pi t}{2}+\frac{\pi}{2}\right) \\ p_2(t)=\begin{cases} p_2(t-1) \text{ if } t>0 \\ p_2(0)=0 \text{ if } t=0 \end{cases} \end{cases}$
	if $(t \bmod 2 \neq 0)$ $\begin{cases} p_1(t)=\sin\left(\frac{k\pi t}{2}+\frac{\pi}{2}\right) \\ p_2(t)=\sin\left(\frac{k\pi(t-1)}{2}+\frac{\pi}{2}\right) \end{cases}$
	$q_i(t) = s_i(t) = 0,\ r_i(t) = 1$

G24_2u	if $(t \bmod 2 = 0)$ $\begin{cases} p_1(t)=\sin\left(\frac{k\pi t}{2}+\frac{\pi}{2}\right) \\ p_2(t)=\begin{cases} p_2(t-1) \text{ if } t>0 \\ p_2(0)=0 \text{ if } t=0 \end{cases} \end{cases}$
	if $(t \bmod 2 \neq 0)$ $\begin{cases} p_1(t)=\sin\left(\frac{k\pi t}{2}+\frac{\pi}{2}\right) \\ p_2(t)=\sin\left(\frac{k\pi(t-1)}{2}+\frac{\pi}{2}\right) \end{cases}$
	$q_i(t) = 0$

G24_3	$p_i(t) = r_i(t) = 1, q_i(t) = s_1(t) = 0$ $s_2(t) = 2 + t \cdot \frac{x_2 \max - x_2 \min}{S}$
G24_3b	$p_1(t) = \sin\left(k\pi t + \frac{\pi}{2}\right),\quad p_2(t) = 1$ $q_i(t) = s_1(t) = 0,\ r_i(t) = 1,$ $s_2(t) = 2 + t \cdot \frac{x_2 \max - x_2 \min}{S}$
G24_3f	$p_i(t) = r_i(t) = 1, q_i(t) = s_1(t) = 0, s_2(t) = 2$
G24_4	$p_2(t) = r_i(t) = 1, q_i(t) = s_1(t) = 0$ $p_1(t) = \sin\left(k\pi t + \frac{\pi}{2}\right), s_2(t) = t \cdot \frac{x_2 \max - x_2 \min}{S}$

G24_5	if $(t \bmod 2 = 0)$ $\begin{cases} p_1(t)=\sin\left(\frac{k\pi t}{2}+\frac{\pi}{2}\right) \\ p_2(t)=\begin{cases} p_2(t-1) \text{ if } t>0 \\ p_2(0) \text{ if } t=0 \end{cases} \end{cases}$
	if $(t \bmod 2 \neq 0)$ $\begin{cases} p_1(t)=\sin\left(\frac{k\pi t}{2}+\frac{\pi}{2}\right) \\ p_2(t)=\sin\left(\frac{k\pi(t-1)}{2}+\frac{\pi}{2}\right) \end{cases}$
	$q_i(t) = s_1(t) = 0,\ r_i(t) = 1,$
	$s_2(t) = t \cdot \frac{x_2 \max - x_2 \min}{S}$

G24_6a/b/c/d	$p_1(t) = \sin\left(\pi t + \frac{\pi}{2}\right), p_2(t) = 1,$ $q_i(t) = s_i(t) = 0, r_i(t) = 1$
G24_7	$p_i(t) = r_i(t) = 1,\ q_i(t) = s_1(t) = 0,$ $s_2(t) = t \cdot \frac{x_2 \max - x_2 \min}{S}$
G24_8a	$p_i(t) = -1, q_1(t) = -(c_1 + r_a \cdot \cos(k\pi t))$ $q_2(t) = -(c_2 + r_a . \sin(k\pi t))$
G24_8b	$p_i(t) = -1, q_1(t) = -(c_1 + r_a \cdot \cos(k\pi t))$ $q_2(t) = -(c_2 + r_a \cdot \sin(k\pi t)), r_i(t) = 1,\ s_i(t) = 0$
k	k determines the severity of function changes, $k = 1$ ~large, $k = 0.5$ ~ medium, $k = 0.25$ ~ small.
S	S determines the severity of constraint changes, $S = 10$ ~large, $S = 20$ ~ medium, $S = 50$ ~ small.
c_1, c_2, r_a (G24_8a/b only)	$c_1 = 1.470561702, c_2 = 3.442094786232,$ $r_a = 0.858958496.$
i	i is the variable index, $i = 1, 2$.

Each dynamic parameter is a time-dependant rule/function which governs the way the problems change.

TABLE IV
PROPERTIES OF EACH TEST PROBLEM IN THE G24 BENCHMARK SET

Problem	ObjFunc	Constr	DFR	SwO	bNAO	OICB	OISB	Path
G24_u	Dynamic	NoC	1	No	No	No	Yes	N/A
G24_1	Dynamic	Fixed	2	Yes	No	Yes	No	N/A
G24_f	Fixed	Fixed	2	No	No	Yes	No	N/A
G24_uf	Fixed	NoC	1	No	No	No	Yes	N/A
G24_2*	Dynamic	Fixed	2	Yes	No	Yes and No	Yes and No	N/A
G24_2u	Dynamic	NoC	1	No	No	No	Yes	N/A
G24_3	Fixed	Dynamic	2-3	No	Yes	Yes	No	N/A
G24_3b	Dynamic	Dynamic	2-3	Yes	No	Yes	No	N/A
G24_3f	Fixed	Fixed	1	No	No	Yes	No	N/A
G24_4	Dynamic	Dynamic	2-3	Yes	No	Yes	No	N/A
G24_5*	Dynamic	Dynamic	2-3	Yes	No	Yes and No	Yes and No	N/A
G24_6a	Dynamic	Fixed	2	Yes	No	No	Yes	Hard
G24_6b	Dynamic	NoC	1	No	No	No	Yes	N/A
G24_6c	Dynamic	Fixed	2	Yes	No	No	Yes	Easy
G24_6d	Dynamic	Fixed	2	Yes	No	No	Yes	Hard
G24_7	Fixed	Dynamic	2	No	No	Yes	No	N/A
G24_8a	Dynamic	NoC	1	No	No	No	No	N/A
G24_8b	Dynamic	Fixed	2	Yes	No	Yes	No	N/A

DFR	Number of disconnected feasible regions
SwO	Switched global optimum between disconnected regions
bNAO	Better newly appear optimum without changing existing ones
OICB	Global optimum is in the constraint boundary
OISB	Global optimum is in the search boundary
Path	Indicate if it is easy or difficult to use mutation to travel between feasible regions
Dynamic	The function is dynamic
Fixed	There is no change
NoC	There is no constraint
*	In some change periods, the landscape either is a plateau or contains infinite number of optima and all optima (including the existing optimum) lie in a line parallel to one of the axes

to achieve better performance. Typical examples are the multipopulation/multiswarm approaches, where multiple subpopulations are used to maintain diversity and each subpopulation/subswarm focuses on tracking one single optimum [26], [27].

B. Chosen Algorithms and Experimental Settings

1) *Chosen Algorithms:* Two commonly used algorithms: triggered hyper-mutation GA (HyperM [21]) and random-immigrant GA (RIGA [24]) were chosen to evaluate the performance of the three strategies mentioned above in DCOPs. HyperM is basically a simple GA with an adaptive mechanism to switch from a low mutation rate (standard-mutation-rate) to a high mutation rate (hyper-mutation-rate, to increase diversity) and vice versa depending on whether or not there is a degradation of the best solution in the population. It represents the "introducing diversity" and "tracking previous optima" strategies in DO.

RIGA is another derivative of a basic GA. After the normal mutation step, a fraction of the population is replaced with randomly generated individuals. This fraction is determined by a random-immigrant rate (also named replacement rate). By continuously replacing a part of the population with random solutions, the algorithm is able to maintain diversity throughout the search process to cope with dynamics. RIGA represents the "maintaining diversity" strategy in DO.

One reason to choose these algorithms for the test is that their strategies are still commonly used in most current state-of-the-art DO algorithms. Another reason is the strategies in these algorithms are very simple and straightforward, making it easy to test and analyze their behavior. In addition, because these two algorithms are very well studied, using them would help in comparing new experimental data with existing results. Finally, because both algorithms are developed from a basic GA (actually the only difference between HyperM/RIGA and a basic GA is the mutation strategy), it would be easier to compare/analyze their performance.

TABLE V
21 TEST CASES (PAIRS) TO BE USED IN THIS PAPER

Static Problems: Unconstrained Versus Fixed Constraints			
1	G24_uf (fF, noC)	versus	G24_f (fF, fC)
Fixed objectives versus dynamic objectives			
2	G24_uf (fF, noC)	versus	G24_u (dF, noC)
3	G24_f (fF, fC, OICB)	versus	G24_1 (dF, fC, OICB)
4	G24_f (fF, fC, OICB)	versus	G24_2 (dF, fC, ONICB)
Dynamic objectives: Unconstrained versus fixed constraints			
5	G24_u (dF, noC)	versus	G24_1 (dF, fC, OICB)
6	G24_2u (dF, noC)	versus	G24_2 (dF, fC, ONICB)
Fixed constraints versus dynamic constraints			
7	G24_1 (dF, fC, OICB)	versus	G24_4 (dF, dC, OICB)
8	G24_2 (dF, fC, ONICB)	versus	G24_5 (dF, dC, ONICB)
9	G24_f (fF, fC)	versus	G24_7 (fF, dC, NNAO)
10	G24_3f (fF, fC)	versus	G24_3 (fF, dC, NAO)
No constraint versus dynamic constraints			
11	G24_u (dF, noC)	versus	G24_4 (dF, dC, OICB)
12	G24_2u (dF, noC)	versus	G24_5 (dF, dC, ONICB)
13	G24_uf (fF, noC)	versus	G24_7 (fF, dC)
Moving constraints expose better optima versus not expose optima			
14	G24_3f (fF, fC)	versus	G24_3 (fF, dC, NAO)
15	G24_3 (fF, dC, NAO)	versus	G24_3b (dF, dC, NAO)
Connected feasible regions versus disconnected feasible regions			
16	G24_6b (1R)	versus	G24_6a (2DR, hard)
17	G24_6b (1R)	versus	G24_6d (2DR, hard)
18	G24_6c (2DR, easy)	versus	G24_6d (2DR, hard)
Optima in constraint boundary versus optima NOT in constr boundary			
19	G24_1 (dF, fC, OICB)	versus	G24_2 (dF, fC, ONICB)
20	G24_4 (dF, dC, OICB)	versus	G24_5 (dF, dC, ONICB)
21	G24_8b (dF, fC, OICB)	versus	G24_8a (dF, noC, ONISB)
dF	dynamic objective func	fF	fixed objective function
dC	dynamic constraints	fC	fixed constraints
OICB	optima in constraint bound	ONICB	opt. not in constraint bound
OISB	optima in search bound	ONISB	optima not in search bound
NAO	better newly appear optima	NNAO	no better newly appear opt
2DR	2 disconn. feasible regions	1R	one single feasible region
Easy	easy for mutation to travel between disconn. regions	Hard	less easy to travel among regions
noC	unconstrained problem	SwO	switched optimum between disconnected regions

The performance of HyperM and RIGA was also compared with a basic GA to see if they work well on the tested problems.[2]

2) *Parameter Settings:* Table VI shows the detailed parameter settings for HyperM, RIGA, and GA. All algorithms use real-valued representations. The algorithms were tested on 18 benchmark problems described in Section III. To create a fair testing environment, the algorithms were tested in a wide range of dynamic settings (different values of population size, severity of change, and frequency of change) with five levels: *small, medium small, medium, medium large, large*.

The evolutionary parameters of all tested algorithms were set to similar values or the best known values if possible. The base mutation rate of the algorithms is 0.15, which is the average value of the best mutation rates commonly used for GA-based algorithms in various existing studies on continuous DO, which are 0.1 ([28], [29]) and 0.2 ([27], [30]). For HyperM and RIGA, the best *hyper-mutation-rate* and *random-immigrant-*

[2]Note that to save space, some tables/figures in this section include not only GA/RIGA/HyperM but also another algorithm: GA+Repair. This algorithm will be introduced in the later sections. This section only focuses on the data relating to GA, RIGA, and HyperM.

TABLE VI
TEST SETTINGS FOR ALL ALGORITHMS USED IN THE PAPER

All algorithms (exceptions below)	Pop size (pop_size)	5, 15, 25 (medium), 50, 100
	Elitism	Elitism and nonelitism if applicable
	Selection method	Nonlinear ranking as in [33]
	Mutation method	Uniform, $P = 0.15$
	Crossover method	Arithmetic, $P = 0.1$
HyperM	Triggered mutate	Uniform, $P = 0.5$ as in [21]
RIGA	Rand-immig. rate	$P = 0.3$ as in [24]
GA+Repair	Search pop size	pop_size$\times$ (4/5)
	Reference pop size	pop_size$\times$ (1/5)
	Replacement rate	0 (default is 0.25 as in [33])
Benchmark problem settings	Number of runs	50
	Number of changes	$5/k$ (see below)
	Change frequency	250, 500, 1000 (med), 2000, 4000 evaluations
	ObjFunc severity k	0.25 (small), 0.5 (med), 1.0 (large)
	Constr. severity S	10 (small), 20 (medium), 50 (large)

rate parameter values observed in the original papers [21], [24] were used. The same implementations as described in [21] and [24] were used to reproduce these two algorithms. A crossover rate of 0.1 was chosen for all algorithms because, according to the analysis in [31], this value was one of the few settings where all tested algorithms perform well on this benchmark set.

A further study of the effect of different values of the base mutation rates, hyper-mutation rates, random-immigrant rates and crossover rates on algorithm performance was also carried out. Detailed experimental results and discussion for this analysis can be found in [31], where it was found that the overall behaviors of the algorithms are not different from those using the default/best known settings, except for the following.

a) When the base mutation rate is very low (≤ 0.01), the performance of GA and HyperM drop significantly.
b) Generally to work well in the tested DCOPs, algorithms need to use high base mutation rates. The range of best mutation rates is 0.3–0.8.
c) Algorithms like RIGA and HyperM also need high random-immigrant/hyper-mutation rates to solve DCOPs. The best results are usually achieved with the rates of 0.6–0.8.
d) The suitable range of crossover rate is 0.1–1.0.

3) *Constraint Handling:* It is necessary to integrate existing DO algorithms with a CH mechanism to use these algorithms for solving DCOPs. That CH mechanism should not interfere with the original DO strategies so that it is possible to correctly evaluate whether the original DO strategies would still be effective in solving DCOPs. To satisfy this requirement, the penalty function approach in [32] was chosen because it is the simplest way to apply existing unconstrained DO algorithms directly to solving DCOPs without changing the algorithms. Also this penalty method can be effective in solving difficult numerical problems without requiring users to choose any penalty factor or other parameters [32].

4) *Performance Measures:* For measuring the performance of the algorithms in this particular experiment, an existing measure: the *modified offline error* [27] was modified. The measure is calculated as the average over, at every evaluation,

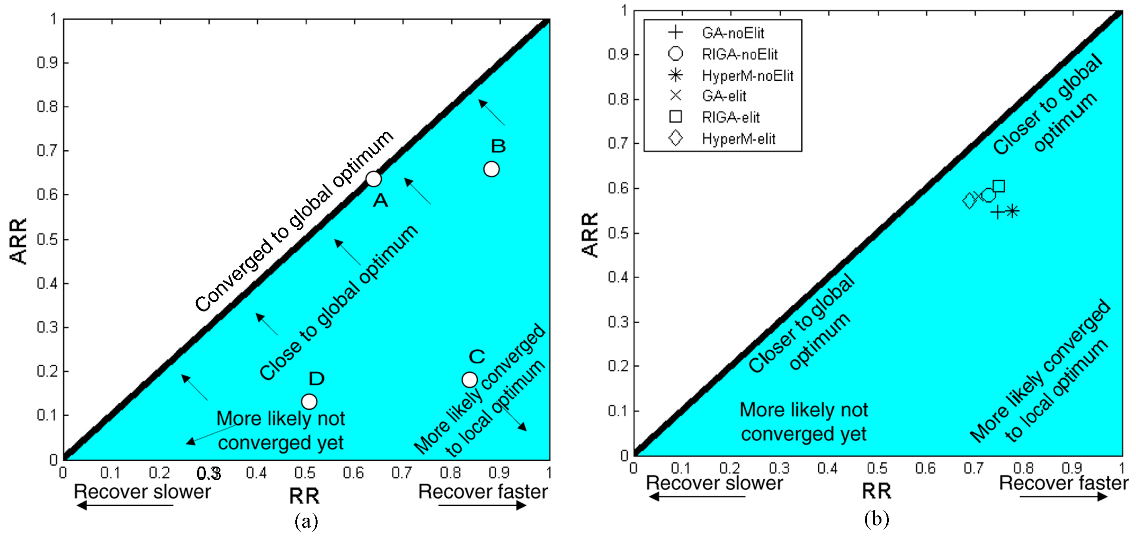

Fig. 1. (a) This provides a guideline for analyzing the convergence behavior/recovery speed of an algorithm given its RR/ARR scores. These scores can be represented as the x and y coordinations of a point on the diagonal thick line or inside the shaded area. The position of the point represents the behavior of the corresponding algorithm. The closer the point is to the right, the faster the algorithm was in recovering and re-converging, and vice versa. In addition, if the point lies on the thick diagonal line (where $RR = ARR$) like point A, the algorithm has been able to recover from the change and converged to the new global optimum. Otherwise, if the point lies inside the shaded area, the algorithm either has converged to a local solution (e.g., point C); or has not been converged yet (e.g., point D: recover slowly, point B: recover quickly). (b) Mapping of the RR/ARR scores of GA, RIGA, and HyperM to the RR-ARR diagram.

the error of the best solution found since the last change of the environment.

Because the measure above is designed for unconstrained environments, it is necessary to modify it to evaluate algorithm performance in constrained environments. At every generation, instead of considering the best errors/fitness values of *any* solutions regardless of feasibility as implemented in the original measure, only the best fitness values/best errors of *feasible* solutions at each generation are considered. If in any generation there is no feasible solution, the measure takes the worst possible value that a feasible solution can have for that particular generation. This measure is called the modified offline error for DCOPs, or offline error for short

$$E_{MO} = \frac{1}{num\_of\_gen} \sum_{j=1}^{num\_of\_gen} e_{MO}(j) \tag{3}$$

where $e_{MO}(j)$ is the best *feasible* error since the last change at the generation j.

Five new measures were also proposed to analyze why a particular algorithm might work well on a particular problem. The first two measures are the recovery rate (RR) and the absolute recovery rate (ARR) to analyze the convergence behavior of algorithms in dynamic environments. The RR measure is used to analyze how quickly an algorithm recovers from an environmental change and starts converging to a new solution before the next change occurs. The new solution is not necessarily the global optimum

$$RR = \frac{1}{m} \sum_{i=1}^{m} \frac{\sum_{j=1}^{p(i)} \left[f_{\text{best}}(i, j) - f_{\text{best}}(i, 1)\right]}{p(i)\left[f_{\text{best}}(i, p(i)) - f_{\text{best}}(i, 1)\right]} \tag{4}$$

where $f_{\text{best}}(i, j)$ is the fitness value of the best feasible solution since the last change found by the tested algorithm until the jth generation of the change period i, m is the number of changes and $p(i), i = 1 : m$ is the number of generations at each change period i. The RR score would be 1 in the best case where the algorithm is able to recover and converge to a solution immediately after a change, and would be close to zero in case the algorithm is unable to recover from the change at all.[3]

The RR measure only indicates if the considered algorithm converges to a solution and if it converges quickly. It does not indicate whether the converged solution is the global optimum. For example, RR can still be 1 if the algorithm does nothing but keep re-evaluating the same solution. Because of that, another measure is needed: the ARR. This measure is very similar to the RR but is used to analyze how quick it is for an algorithm to start converging to the global optimum before the next change occurs

$$ARR = \frac{1}{m} \sum_{i=1}^{m} \frac{\sum_{j=1}^{p(i)} \left[f_{\text{best}}(i, j) - f_{\text{best}}(i, 1)\right]}{p(i)\left[f^*(i) - f_{\text{best}}(i, 1)\right]} \tag{5}$$

where $f_{\text{best}}(i, j), i, j, m, p(i)$ are the same as in (4) and $f^*(i)$ is the global optimal value of the search space at the ith change. The ARR score would be 1 in the best case when the algorithm is able to recover and converge to the global optimum immediately after a change, and would be zero in case the algorithm is unable to recover from the change at all. Note that the score of ARR should always be less than or equal to that of RR. In the ideal case (converged to global optimum), ARR should be equal to RR.[4]

[3]Note that RR will never be equal to zero because there is at least one generation where $f_{\text{best}}(i, j) = f_{\text{best}}(i, p(i))$.

[4]Note that to use the measure ARR it is necessary to know the global optimum value at each change period.

The RR and ARR measures can be used together to indicate if an algorithm is able to converge to the global optimum within the given time frame between changes and if so how quickly it converges. The *RR-ARR diagram* in Fig. 1 shows some analysis guidelines.

A third measure, percentage of selected infeasible individuals, is proposed to analyze algorithm ability to balance exploiting feasible regions and exploring infeasible regions in DCOPs. This measure finds the percent of infeasible individuals selected for the next generation. The average (over all tested generations) is then compared with the percentage of infeasible areas in the search space. If the considered algorithm is able to accept *infeasible* diversified individuals in the same way as it accepts *feasible* diversified individuals (and hence to maintain diversity effectively), the two percentage values should be equal.

To analyze the behavior of algorithms using triggered-mutation mechanisms such as HyperM, a fourth measure: triggered-time count, which counts the number of times the hyper-mutation-rate is triggered by the algorithm, and a fifth measure: detected-change count, which counts the number of triggers actually associated with a change, are also proposed. For HyperM, triggers associated with a change are those that are invoked by the algorithm within ν generations after a change, where ν is the maximum number of generations (five in this implementation) needed for HyperM to detect a drop in performance. These two measures indicate how many times an algorithm triggers its hyper-mutation; whether each trigger time corresponds to a new change; and if there is any change that goes undetected during the search process.

Note that the five measures above are all needed for our analysis because they are used to investigate different aspects of the algorithms. Furthermore, all of the measures used here are specifically designed for dynamic problems. This creates a problem for the experiments in this paper because in the G24 benchmark set there are not only dynamic problems but also stationary problems. To overcome this issue, in this paper stationary problems are considered a special type of dynamic problem which still have "changes" with the same change frequency as other dynamic problems. However, in stationary problems the changes do not alter the search space.

C. Experimental Results and Analyses

The full *offline-error* results of the tested algorithms on all 18 benchmark problems for all test scenarios are presented in the tables in [34]. These data were further analyzed from different perspectives to achieve a better understanding of how existing DO strategies work in DCOPs and how each characteristic of DCOPs would affect the performance of existing DO algorithms. First, the average performance of the tested algorithms on each major group of problems under different parameter settings and dynamic ranges were summarized to have an overall picture of algorithm behavior on different types of problems (see Fig. 2). Then the effect of each problem characteristic on each algorithm was analyzed in 21 test cases (each case is a pair of almost identical problems, one with a particular characteristic and one without) as shown in Table V of Section III (see the test results in Figs. 3, 4). For

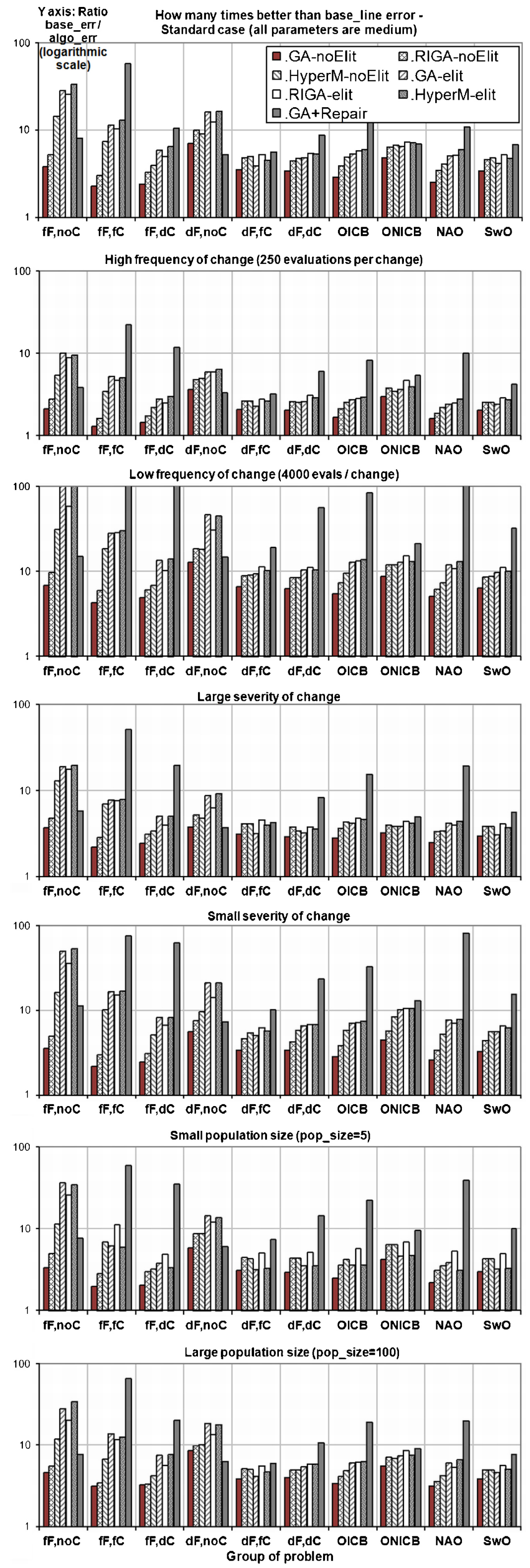

Fig. 2. Algorithm performance in groups of problems. Performance (vertical axis in logarithmic scale) is evaluated by calculating the ratio between the *base line* (worst error among all scenarios) and the error of each algorithm in each problem to see how many times their performance is better (smaller) than the *base line*. Explanations for abbreviations can be found in Table V.

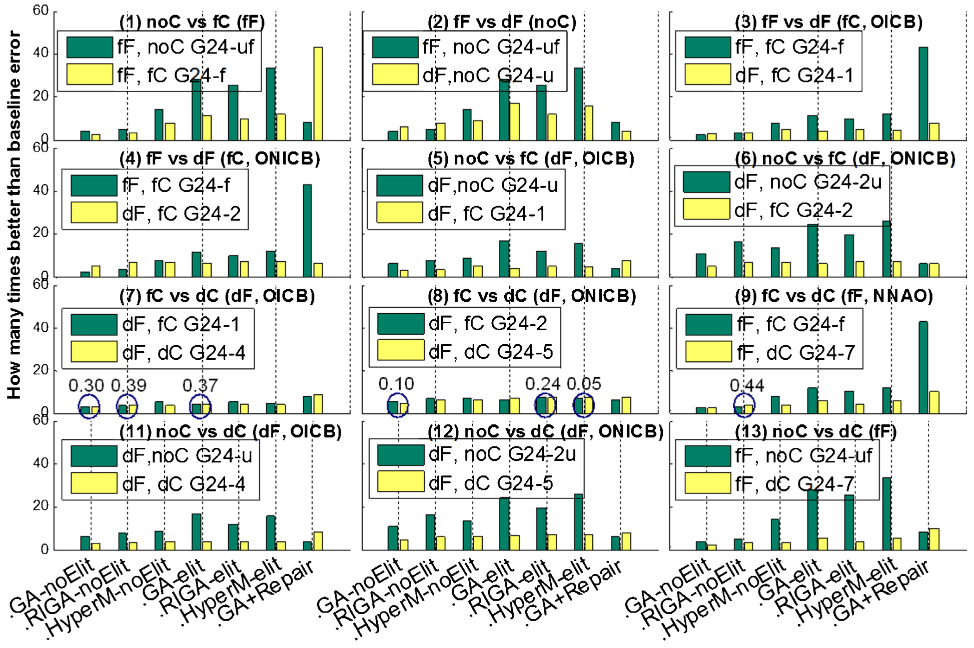

Fig. 3. Effect of 12 different problem characteristics on algorithm performance (medium case). Performance (vertical axis) is evaluated based on the ratio between the base line error (described in Fig. 2) and algorithm errors. Each subplot represents algorithm performance (pair of adjacent bars) in a pair of almost identical problems (one has a special characteristic and the other does not). The larger the difference between the bar heights, the greater the impact of the corresponding DCOP characteristic on performance. Subplots' title represents the test case numbers (in brackets) followed by an abbreviated description. Explanations for the abbreviations are in the last rows of Table V. Pairs where the impact of a characteristic on an algorithm is *not significant* (according to a t-test with significance level of 0.05) are circled and in such cases the t-test scores are also given to highlight the level of insignificance.

each particular algorithm, some further analyses were also carried out using the five newly proposed measures mentioned above. Details of these analyses will be described in the next subsections. Only the summarized results are presented in Fig. 2 with different settings (small/medium/large). For other detailed figures and tables, the results will only be presented in the default settings (all parameters and dynamic range are set to medium). For detailed results in other settings, readers are referred to [34].

A statistical t-test with a significance level of 0.05 was done to evaluate the level of significance of the possible impacts that each characteristic of DCOPs can have on the performance of the tested algorithms.[5] The summarized results of this statistical test can be found in Figs. 3 and 4.

The experimental results show some interesting, and in some cases, surprising findings.

1) *Impact of Different Dynamic Ranges on Algorithm performance:* The summarized results in groups of problems (Fig. 2) show that: 1) generally the behavior of algorithms and their relative strengths/weaknesses in comparison with other algorithms still remain roughly the same when the dynamic settings change; and 2) as expected in most cases algorithms' performance decreases when the conditions become more difficult (magnitude of change becomes larger, change frequency becomes higher, population size becomes much smaller). Among the variations in dynamic settings, it seems that the variations in frequency of change affect algorithms' performance the most, followed by variations in magnitude of changes. Variations in population size have the least impact on algorithm performance.

2) *Effect of Elitism on Algorithm Performance:* The summarized results in groups of problems (Fig. 2) and the pair-wise comparisons in Figs. 3 and 4 reveal an interesting effect of elitism on both unconstrained and constrained dynamic cases: the elitism versions of GA/RIGA/HyperM perform better than their nonelitism counterparts in most tested problems. The reason for this effect (with evidence shown in the next paragraph) is that elitism helps algorithms with diversity-maintaining strategies to converge faster. This effect is independent of the combined CH techniques.

Two measures proposed in Section IV-B4, RR and ARR, were used to study the inefficiency of GA/RIGA/HyperM in the nonelitism case. The scores of the algorithms on these measures are given in Fig. 1(b). The figure shows that none of the algorithms are close to the optimum line, meaning there are problems/change periods where the algorithms were unable to converge to the global optimum. In addition, for RIGA, its elitism version is closer to the top-right corner while its nonelitism version is closer to the bottom-left corner, meaning that nonelitism makes RIGA converge slower/less accurately. Finally, for GA/HyperM, their elitism versions are closer to the global optimum while their nonelitism versions are closer to the bottom-right corner, meaning that the nonelitism versions of GA/HyperM are more susceptible to premature convergence. The results hence show that the high diversity maintained by the random-immigrant rate in RIGA and the high mutation rate in GA/HyperM come with a tradeoff: the convergence speed is affected. In such a situation, elitism can

[5] t-test is considered robust under the conditions of this experiment [35, ch. 37].

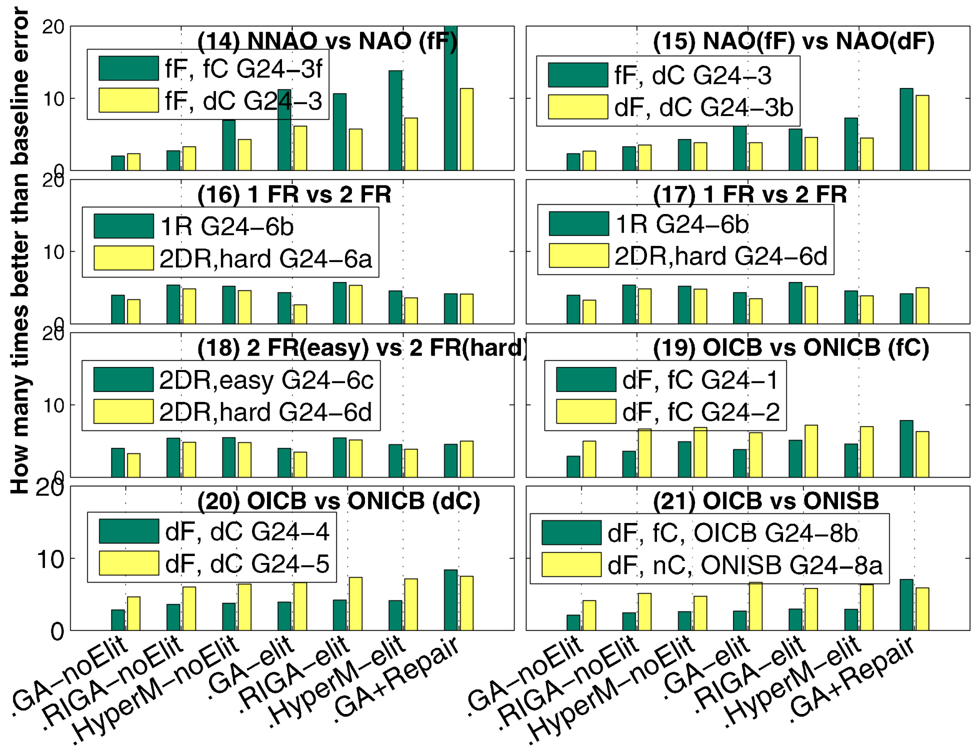

Fig. 4. Effect of the other eight different problem properties on algorithm performance (medium case). Instructions to read this figure can be found in Fig. 3. All eight characteristics have statistically significant impacts on the algorithms, and hence there is no bar with circles.

be used to speed up the convergence process. Elite members can guide the population to exploit the good regions faster while still maintaining diversity.

3) *Effect of Infeasible Areas on Maintaining/Introducing Diversity:* Another interesting observation is that the presence of constraints makes the performance of diversity-maintaining/introducing strategies less effective when used in combination with the tested penalty functions. This behavior can be seen in Fig. 2 where the performance of all algorithms in the unconstrained dynamic case (dF+noC) is significantly better than their performance in all dynamic constrained cases (dF+fC, fF+dC, dF+dC). This behavior can also be seen in the more accurate pair-wise comparisons in Figs. 3 and 4: for each pair of problems in which one has constraints and the other does not, GA, RIGA and HyperM always perform worse on the problem with constraints (see pairs 1, 5, 6, 11, 12, 13 in Fig. 3, pair 21 in Fig. 4).

The reason for this inefficiency is the use of tested penalty functions prevents diversity-maintaining/introducing mechanisms from working effectively. In solving unconstrained dynamic problems, all diversified individuals generated by the diversity maintaining/introducing strategies are useful because they contribute to either: 1) detecting newly appearing optima; or 2) finding the new place of the moving optima. In DCOPs, however, there are two difficulties that prevent diversified individuals that are infeasible from being useful in existing DO strategies. One difficulty is many diversified but infeasible individuals might not be selected for the next generation population because they are penalized with lower fitness values by the penalty functions. Consequently, these diversified individuals cannot be used for maintaining diversity

TABLE VII
AVERAGE *Percentage of Selected Infeasible Individuals* OVER 18 PROBLEMS

Algorithms	Percent of infeasible solutions
.GA-elit	23.0%
.RIGA-elit	37.6%
.HyperM-elit	26.4%
.GA-noElit	46.3%
.RIGA-noElit	49.1%
.HyperM-noElit	45.3%
Percentage of infeasible areas	**60.8%**

The last row shows the average *percentage of infeasible areas*.

unless they are re-introduced again in the next generation. To demonstrate this drawback, the previously proposed measure percentage of selected infeasible individuals was used. As can be seen in Table VII, in the elitism case the percentage of infeasible solutions in the population (23–37.6%) is much smaller than the percentage of infeasible areas over the total search space (60.8%). This means only a few of the diversified, infeasible solutions are retained and hence the algorithms are not able to maintain diversity in the infeasible regions.[6]

The second difficulty is that, even if a diversified but infeasible individual is selected for the next generation, it might no longer have its true fitness value. Consequently, environmental changes might not be accurately detected or tracked.

[6]Nonelitism algorithms are able to retain more infeasible individuals, of which some might be diversified solutions. However, as shown in Section IV-C2, in the nonelitism case this higher percentage of infeasible individuals comes with a tradeoff of slower/less accurate convergence, which leads to the generally poorer performance.

TABLE VIII

Triggered-Time Count SCORES AND THE *Detected-Change Count* SCORES OF HYPERM IN A PAIR OF PROBLEMS WITH MOVING CONSTRAINTS EXPOSING NEW OPTIMA AFTER 11 CHANGES

Algorithms	G24_3 (NAO+fF)				G24_3b (NAO+dF)			
	Trigger Count		Detected Change Count		Trigger Count		Detected Change Count	
	Value	stdDev	Value	stdDev	Value	stdDev	Value	stdDev
HyperM-noElit	188.70	8.40	1.74	0.78	199.83	5.88	11.00	0.00
HyperM-Elit	0.00	0.00	0.00	0.00	30.43	0.57	11.00	0.00

NAO: newly appearing optimum.
fF/dF: fixed/dynamic objective function.

4) *Effect of Switching Global Optima (Between Disconnected Feasible Regions) on Strategies That Use Penalty Functions:* The results show existing DO methods become less effective when they are used in combination with the tested penalty functions to solve a special class of DCOPs: problems with disconnected feasible regions where the global optimum switches from one region to another whenever a change occurs. In addition, the more separated the disconnected regions are, the more difficult it is for algorithms using penalty functions to solve.

The reason for this difficulty is the necessity to have a path through the infeasible areas that separate the disconnected regions to track the moving optimum. This path might not be available if penalty functions are used because penalties make it unlikely infeasible individuals are accepted. Obviously the larger the infeasible areas between disconnected regions, the harder it is to establish the path using penalty methods.

Three test cases (pairs of almost identical problems) 16, 17, 18 in Table V were used to verify the statement above. In all three test cases the objective functions are the same and the global optimum switches between two locations whenever a change occurs. However, each case represents a different dynamic situation. Case 16 tests the situation where in one problem of the pair (G24_6b) there is a feasible path connecting the two locations and in the other problem (G24_6a) the path is infeasible, i.e., there is an infeasible area separating two feasible regions. Case 17 is the same as case 16 except that the infeasible area separating two feasible regions has a different shape. Case 18 tests a different situation where in one problem (G24_6c) the infeasible area separating the two feasible regions is small whereas in the other problem (G24_6d) this infeasible area is large.

The experimental results in these three test cases (pairs 16, 17, 18 in Fig. 4) confirm the hypotheses stated in the beginning of this subsection. In cases 16 and 17, the performance of the tested algorithms did decrease when the path between the two regions is infeasible. In case 18, the larger the infeasible area separating the two regions, the worse the performance of the tested algorithms.

5) *Effect of Moving Infeasible Areas on Strategies That Track the Previous Optima:* Algorithms relying on tracking previous global optimum such as HyperM might become less effective when the moving constraints expose new, better optima without changing the existing optima. The reason is HyperM cannot detect changes in such DCOPs and hence might not be able to trigger its hyper-mutation rate. With the currently chosen base mutation of 0.15, HyperM is still able to produce good results because the mutation is high enough for the algorithm to maintain diversity. However, in a previous study [17], when a much smaller base mutation rate was used, HyperM becomes significantly worse compared to other algorithms in solving problems like G24_3.

To illustrate this drawback, the newly proposed measures *triggered-time count* and *detected-change count* were used to analyze how the triggered-hypermutation mechanism works on problem G24_3. As can be seen in Table VIII, HyperM either was not able to trigger its hyper-mutation rate to deal with changes (elitism case, *triggered-time count*=0 and *detected-change count*=0) or was not able to trigger its hyper-mutation rate correctly when a change occurs (nonelitism case, *triggered-time count*$\sim$188.7 and *detected-change count*$\sim$1.74). It is worth noting in the nonelitism case, most of the trigger times are caused by the selection process because in nonelitism selection the best solution in the population is not always selected for the next generation.

Table VIII also shows that in problem G24_3b, which is almost identical to G24_3 except it has its existing optima changed, HyperM was able to detect changes and hence trigger its hyper-mutation timely whenever a change occurs. It shows HyperM only becomes less effective where environmental changes do not change the value of existing optima.

D. Possible Suggestions to Improve Current Dynamic Optimization Strategies in Solving DCOPs

The experimental results suggest some directions for addressing the drawbacks listed in the previous subsections.

1) Based on the observation that elitism is useful for diversity-maintaining strategies in solving DCOPs, it might be useful to develop algorithms that support both elitism and diversity maintaining mechanisms.
2) Given that methods like HyperM are not able to detect changes because they mainly use change detectors (the best solution in case of HyperM) in the feasible regions, it might be useful to use change detectors in both regions and infeasible regions.
3) Because experimental results show that tracking the existing optima might not be effective in certain cases of DCOPs, it might be useful to track the moving feasible regions instead. Because after a change in DCOPs the global optimum always either moves along with the

feasible areas or appears in a new feasible area, an algorithm able to track feasible areas would have higher chance of tracking the actual global optimum.

V. Challenges of Some Constraint Handling Strategies in Solving DCOPs

A. *Difficulties in Handling Dynamics*

The most obvious reason for the difficulties in applying existing CH strategies to solving DCOPs is these strategies are not designed to handle environmental dynamics. One might then question whether these difficulties can be overcome by combining existing CH strategies with existing DO strategies.

Unfortunately, as will be shown below, not all difficulties can be resolved by combining existing CH strategies with existing DO strategies. In addition, this combination might also bring some new challenges due to the conflict of the optimization goals of the two types of strategies. These are the challenges in maintaining diversity, introducing diversity, and detecting changes based on performance drop.

1) *Impacts on Maintaining/Introducing Diversity:* As already discussed, one of the important strategies in DO is to maintain/introduce diversity in the whole search space to detect changes and to find newly-appearing/moving optima. However, diversity might no longer be maintained this way when combined with some CH techniques.

In many CH techniques, the original space is specifically transformed so algorithms only focus on certain areas instead of the whole original space. In such cases, even if a diversity-introducing strategy such as HyperM is used to generate individuals in the whole search space, diversified individuals generated in the unfocused areas might be neglected by the algorithms and hence do not contribute to maintaining diversity. Typical examples of CH strategies that adopt this search space transformation approach are penalty methods where the constrained search space is transformed to an unconstrained search space with penalized fitness values. Another example is some approaches use special representations/operators. In these approaches, the algorithms might be restricted to searching only in the feasible regions, in a transformed feasible search space, or in the boundaries of feasible regions. Detailed reviews/references for representative penalty approaches and special representations/operators approaches can be found in [36] and [37].

In some other CH techniques, individuals are selected not exclusively based on their actual fitness values but also on some special specifications. For example, in stochastic ranking [38] infeasible individuals might have a better chance of being accepted based on the given stochastic parameter. A contrary example can be found in simple multimembered ES [39] where infeasible solutions are less likely to be accepted even if they have higher fitness values than the feasible ones. Another example is in a CH multiobjective approach [40] where individuals are ranked not entirely based on their original fitness but also on the number of violated constraints. In CH techniques like these, diversified individuals generated by DO strategies might not be selected in the same way as they were originally designed for, i.e., the number of infeasible diversified individuals might become too large or too small. The way a diversity maintaining strategy works might not be the same as in the unconstrained case.

Experimental evidence for the inefficiency mentioned above has already been shown in Section IV-C, where the diversity-maintaining/introducing strategies become less effective when combined with the tested penalty methods.

In [31], it was shown that the current state-of-the-art in CH such as SRES [38], [41] and SMES [39] become much less effective in DCOPs and could not maintain enough diversity to deal with the dynamics in DCOPs.

2) *Impacts on Change Detection:* Another possible difficulty of combining CH strategies with DO strategies is the use of some existing CH techniques might make change detection based on performance drop, a common DO technique, less effective. As already mentioned in Section IV-B, algorithms like HyperM assume that during the search process, if there is a degradation in the fitness values of the best solution found in each generation, there might be a change in the search space. However, when DO algorithms are combined with some CH techniques to solve DCOPs, such degradation in best fitness values might no longer be caused by an actual change in the search space. Instead, the degradation might be caused either by an increase in penalty values or by the elimination of the current good solutions from the population.

One example can be found in some CH techniques such as dynamic penalty or adaptive penalty [42]–[44], where the degradation of (modified) fitness values is not caused by environmental changes but by the increase over time of the penalty values. The consequence of this dynamic/adaptive scheme is that if the detector solutions used by the change detection method are infeasible or become infeasible, over time their fitness value will decrease.[7]

In some other CH techniques which use ranking-based methods [38]–[40], during the selection process the current better solutions might be dropped in favor of other solutions, which might have worse fitness values but are more useful for the CH process. In these situations there might also be a drop in the values of the best solutions at each generation.

The drop in fitness values of the detector solutions in both cases above might be incorrectly considered by DO strategies like HyperM to be a change in the environment and this might consequently trigger the DO strategies to react inappropriately.

B. *Difficulties in Handling Constraints (Empirical Evidence Shown in Section V-D)*

The difficulties of applying some existing CH strategies to solving DCOPs are also caused by that their CH ability becomes less effective. This is due to two reasons.

1) *Issue of Outdated Information:* In DOPs, after a change, all existing information that an algorithm has acquired or has been given about the problem might become outdated and

[7]Of course, in penalty methods, if change detections are made on the original fitness instead of on the penalized fitness, the increase of penalty values will not have any impact on detecting changes. However, in this case, change detection might suffer from another problem: changes in constraint functions will go undetected unless additional improvements are made to detect constraint changes explicitly.

consequently make the algorithm less effective. For example, in algorithms using strictly feasible reference individuals like Genocop III [33], [45], after a change some reference individuals may become infeasible. Similarly, in some "decoder" methods the reference lists for ordinal representations [e.g.. the ordered lists of cities (TSP [46])/ordered lists of Knapsack items (KSP [47])/order lists of tasks (scheduling [48])] might no longer be in order after a change because the cities/items/tasks have changed their values. Another example can be found in dynamic/adaptive penalty methods (e.g., [43], [44]) where the penalty parameters learnt by the methods might no longer be suitable because the balance between feasible and infeasible solutions has changed.

2) *Issue of Outdated Strategy:* The CH strategies themselves can also be outdated when solving DCOPs. This might occur when the CH strategies have problem-dependent parameters, whose values might be tailored to work best in only one (class of) stationary environment, to solve a DCOP. In such cases, if the parameters are fine-tuned for the problem before change, the algorithm might only work well until a change occurs. Typical examples are penalty methods with pre-defined penalty factors and/or other pre-defined parameters that control how the penalty is defined. Other examples are some combinatorial repair methods, methods with special operators, or decoder methods. Detailed reviews are in [37] and [49].

Strategy-being-outdated might also occur with many adaptive CH strategies that are not problem-dependent because these strategies rely on some specific assumptions that are only true in stationary problems.

Typical examples are self-adaptive fitness formulation [50] and stochastic ranking [38]. The general approach of these strategies is to balance feasibility/infeasibility based on the performance of the current population, assuming that the population always reflects a "memory" of information about the search space and the convergence process. This assumption is not true in dynamic environments. When a change occurs, the search space might change its shape and consequently the "memory" of the population no longer reflects the property of the new search space but only a small area where the population currently is. This disadvantage has been observed in [31] for the case of the state-of-the-art SRES.

Another type of CH strategies relying on outdated assumptions are dynamic/adaptive methods that use the running time value (e.g., the number of generations so far) to balance feasibility and infeasibility. CH strategies of this type [39], [42], [43], [51], [52] assume that the population will eventually converge to the good regions and hence they handle constraints by increasingly rejecting more infeasible solutions when time goes by, or by reducing the mutation step size when time goes by, to increase the convergence speed to good regions. In DCOPs, because after a change good feasible regions might no longer be good or feasible, if the CH strategy still imposes its previous balancing mechanism to increase convergence speed, the algorithm could end up converging to the wrong place and will not be able to track the moving optima. This disadvantage has been experimentally confirmed in [31] for the case of the state-of-the-art SMES.

C. Possible Suggestions to Improve Current Constraint Handling Strategies in Solving DCOPs

The discussions in the two previous subsections show that, to handle constraints effectively in DCOPs, a CH strategy might need to satisfy the requirements below.

1) Make sure that the goal of CH does not conflict with the goal of DO. Particularly:
   a) allow diversified individuals to be distributed in the whole search space;
   b) do not reject diversified individuals even if they do not contribute to CH;
   c) pay special attention whenever changes are detected by monitoring the fitness values of current individuals (it is necessary to check to see if a drop in performance is really caused by an environmental change).
2) Make sure that the algorithm is updated whenever a change occurs. Particularly:
   a) problem knowledge needs to be updated;
   b) the CH strategy might also need to be updated whenever a change occurs.

An algorithm needs to handle both environmental dynamics and constraints effectively to work well in DCOPs. This means that a "good" algorithm for DCOPs needs to satisfy not only the requirements for CH above but also the four requirements for DO identified in Section IV-D.

D. Experimental Analyses

An experimental analysis was carried out to test the performance of the *repair method*, a representative CH strategy, on the G24 benchmark set. The purpose is to answer three questions: 1) what is the usefulness of the repair method in solving DCOPs; 2) whether the hypothesis about the difficulties of DCOPs toward CH strategies, as mentioned in Section V-B, is true; and 3) if the hypothesis is true, would these difficulties affect the performance of CH strategies (in particular the repair method) in solving DCOPs. These results would help gain more understanding about how to design better algorithms to solve DCOPs.

1) *Chosen Constraint Handling Technique for the Analysis:* For this analysis the *repair method* [33] was chosen because it is representative, simple, easy to implement, problem-independent and is designed specifically for the continuous domain.

Repair-based methods, however, also have one disadvantage: they may require a considerable number of feasibility checks to find a feasible individual. As a result, repair-based methods might not be suitable for solving problems with very expensive constraint functions and problems with very small feasible areas.

2) *Repair Algorithms and the Method in Genocop III [33]:* a) *General ideas:* The idea of repairing is, if it is possible to map (repair) an infeasible solution to a feasible solution, then instead of searching the best feasible solution directly, it might be possible to look for an individual that can potentially produce the best repaired solution. The better the repaired solution, the higher the fitness value of an individual. In certain

cases, the feasible solution created by the repair process can also be used to replace some of the search individuals.

Generally, a repair process can be described in three steps.

1) If a newly created individual **s** (can be feasible or infeasible) needs repair, use a heuristic *repair* () to repair **s**, mapping **s** to a new, feasible individual **z**.
2) The objective value $f(\mathbf{z})$ of **z** is used as input to calculate the fitness value of **s**, $\text{eval}(\mathbf{s}) = h(f(\mathbf{z}))$ where h is the mapping from objective values to fitness.
3) If the repair approach is Lamarckian, replace one or more search individuals by **z**.

In the *repair method* [33], [45] chosen for this experiment, the *repair* () heuristic is as follows.

1) The population is divided into two subpopulations: a search population S containing normally-evolving individuals, which can be fully feasible or only linearly feasible, and a reference population R containing only fully feasible individuals.
2) During the search process, while each individual **r** in R is evaluated using their objective function as usual, each individual **s** in S is considered to be repaired based on an individual from R. Details of the repair routine can be found in Algorithm 1.

It is important to note there are two possible variants of deciding whether a search individual **s** needs to be repaired in Genocop III (step 2 above). In the first variant [45], a search individual **s** is repaired *only* if **s** is infeasible. In the second and latest variant [33], the implementation shows that search individuals are repaired regardless of their feasibility.

In all experiments in this paper, the second variant was implemented. From now on, unless stated otherwise the term *repair method* will be used to refer to the continuous-based repair approach proposed in [33].

b) *Feasibility/infeasibility balancing strategy and problem knowledge in the repair method:* The repair method and other repair approaches have the ability to adaptively balance feasibility and infeasibility. This balance is achieved by accepting both infeasible and feasible individuals, provided that they can produce good repaired solutions and by updating the fitness values of search individuals with those of the mapped, feasible solutions. This way the repair method ensures that infeasible solutions are accepted and they cannot have better fitness values than the best feasible solution available.

The strategy above needs certain problem information, which is provided by the reference population R and the search population S. R is an essential source of information to direct the algorithm toward promising feasible regions [during the repair process (*Repair* routine, Algorithm 1), newly repaired solutions are always generated in the directions toward reference individuals]. R also provides the balancing strategy with information about the best feasible solution available (via their fitness values) so that the strategy can make sure that no infeasible individual can have better fitness values than this best feasible solution.

The search population S is also an essential source of problem information. It helps indicate which point in the search space would lead to potentially promising feasible regions (via repair). In the selection phase the balancing strategy then uses this information to select those individuals that would potentially lead to the most promising regions.

Algorithm 1 *Routine* Repair(Indiv **s**)

```
1) Randomly pick an individual $\mathbf{r} \in R$
2) Generate individual $\mathbf{z}$ in the segment between $\mathbf{s}$ and $\mathbf{r}$
    a) $a = U(0, 1)$
    b) $\mathbf{z} = a.\mathbf{s} + (1 - a).\mathbf{r}$
    c) While $\mathbf{z}$ is infeasible, back to step 2a
    d) If a feasible $z$ is not found after 100 trials, $z = r$ and
       $\text{eval}(\mathbf{z}) = \text{eval}(\mathbf{r})$
3)  a) Evaluate $\mathbf{z}$
    b) If ($f(\mathbf{z})$ better than $f(\mathbf{r})$): $\mathbf{r} = \mathbf{z}$; $\text{eval}(\mathbf{r}) = f(\mathbf{z})$
    c) Update the fitness value of $\mathbf{s}$: $\text{eval}(\mathbf{s}) = f(\mathbf{z})$
4) Return the individual $\mathbf{s}$
```

c) *How can the characteristics of DCOPs affect the repair method?:* The repair method suffers from the problem of outdated information, which in turn makes the feasibility/infeasibility balancing strategy outdated.

The first type of information might become outdated when a change occurs is the fitness values of search individuals. Because the fitness of a search individual is always based on the objective value of the corresponding mapped feasible solution, it is assumed that the search population always offers a "memory" of good areas in the search space and directions toward these good areas. The higher the fitness value of an individual, the better the feasible region achieved by repairing this individual.

In a dynamic environment, the memory, or fitness values of search individuals, can become outdated right after a change if the objective values of the corresponding repaired solutions change. Particularly, the high fitness values of existing individuals might no longer lead to good repaired solutions and vice versa. Worse, search individuals with high-but-outdated fitness values might incorrectly bias the selection process, which makes the search process less effective.

The second type of information that might become outdated when a change occurs is the set of reference individuals that are used to repair all other search individuals. The key assumption that all reference individuals are feasible and are the best in the population is only true in stationary environments. In dynamic environments, after a change, some existing reference individuals might no longer remain the best in the population or might even become infeasible. These outdated reference individuals not only violate the assumption named above but might also wrongly bias the search and drive more individuals *away* from the good regions, making the search process less effective.

In the following experiments an analysis was made to see if the above hypotheses are correct and how significant their effects are.

3) *Experimental Settings:*

a) *Tested algorithms:* In this experiment, the repair method was integrated with a basic GA. The integrated version is called GA+Repair and is described in Algorithm 2. This integration makes it possible to analyze the strengths and weaknesses of the repair strategy because the only difference between GA and GA+Repair is the repair operator and hence any difference in performance would be caused by the repair operator.[8] In addition, because all other tested strategies are integrated with a basic GA, it is natural to integrate the repair method with the GA[9] to compare it with these strategies.

Even though the GA+Repair is a simplified version of Genocop III, both algorithms have very similar behaviors when solving different groups of DCOPs. This similarity suggests the result tested with GA+Repair can be generalized to other approaches that use the repair method. For detailed results of Genocop III's performance in the G24 benchmark set and a comparison of its performance with other existing and new algorithms, readers are referred to the study in [31].

b) *Parameter settings:* The tested algorithms use the same parameter settings as the previously tested GA, RIGA, and HyperM except that the population now is divided into a search population and a reference population (see Table VI), as implemented in the original Genocop III [33].

c) *Performance measures:* Three different measures were used. The first measure, which is the modified version of the *off-line error* measure (see Section IV-B4), was used to evaluate/compare the general performance of the GA+Repair. Similar to the previous experiment, using this measure the average performance of GA+Repair was also summarized in each major group of problems (see results in Fig. 2) and the effect of each problem characteristic on GA+Repair was analyzed in 21 test cases shown in Table V of Section III (see results in Figs. 3, 4).

The second and third measures were specifically proposed for this experiment. The second measure, named feasible reference individuals, was used to analyze the behavior of the *repair method* when some reference individuals become outdated due to environmental changes (see Fig. 5). The third measure, named feasible individuals in each disconnected region, was used to analyze the ability of repair methods to balance feasibility and infeasibility on problems with optima switching between disconnected feasible regions (see Fig. 6). Details of these two measures will be described later.

4) *Impact of Outdated Information/Strategy on the Performance of the Repair Method:*

a) *Overall observation of performance in groups of problems:* In the group of stationary constrained problems (fF, fC), the results in Fig. 2 show that, as expected, a specialized CH technique such as the repair method in GA+Repair

Algorithm 2 GA+Repair

```
Note: It is assumed that the problem is maximization
1) Initialize:
   a) Randomly initialize $m$ individuals in search pop $S$
   b) Initialize $n$ individuals in the reference population $R$
      i) Randomly generate points until a feasible $\mathbf{r}$ is found
      ii) Update fitness: $\mathrm{eval}(\mathbf{r}) = f(\mathbf{r})$ & add $r$ to $R$
2) Search: For $i = 1 : m$
   a) $p_1 = U(0,1)$; $p_2 = U(0,1)$
   b) Crossover: If $(p_1 < P_{Xover})$
      i) Use nonlinear ranking selection to choose a pair of
         parents from $S$
      ii) Crossover an offspring $\mathbf{s}$ from the chosen parents
      iii) Evaluate $\mathbf{s}$ and repair $\mathbf{s}$ using $Repair(\mathbf{s})$
      iv) Use nonlinear ranking selection to replace one of the
          worst individuals in $S$ by $\mathbf{s}$
   c) Mutation: If $(p_2 < P_{Mutate})$
      i) Use nonlinear ranking-selection to choose a parent
         from $S$
      ii) Mutate an offspring $\mathbf{s}$ from the chosen parent
      iii) Evaluate $\mathbf{s}$ and repair $\mathbf{s}$ using $Repair(\mathbf{s})$
      iv) Use nonlinear ranking selection to replace one of the
          worst individuals in $S$ by $\mathbf{s}$
   d) Otherwise: If $(p_1 \geq P_{Xover})$ and $(p_2 \geq P_{Mutate})$
      i) Use nonlinear ranking-selection to choose an individual
         $\mathbf{s}$ from $S$
      ii) If $\mathbf{s}$ has not been evaluated since last generation, eval-
          uate $\mathbf{s}$
      iii) Repair $\mathbf{s}$ using the routine $Repair(\mathbf{s})$
      iv) Using nonlinear ranking selection to replace one of the
          worst individuals in $S$ by $\mathbf{s}$
3) Evolve the reference population after each 100 evaluations:
   For $i = 1 : n$
   a) Crossover: If $(U(0,1) < P_{Xover})$
      i) Use nonlinear ranking-selection to choose a pair of
         parents from $R$, and crossover an offspring $\mathbf{r}$
      ii) If $\mathbf{r}$ is feasible
         A) Evaluate $\mathbf{r}$ and $\mathbf{x}$, the better of the two parents
         B) If $f(\mathbf{r})$ better than $f(\mathbf{x})$ then $\mathbf{x} = \mathbf{r}$ and fitness value
            $\mathrm{eval}(\mathbf{x}) = f(\mathbf{r})$
   b) Mutation: If $(U(0,1) < P_{Mutation})$
      i) Nonlinear ranking-selection to choose a parent $\mathbf{x}$ from
         $R$, and mutate an offspring $\mathbf{r}$ from $\mathbf{x}$
      ii) If $\mathbf{r}$ is feasible
         A) Evaluate $\mathbf{r}$ and $\mathbf{x}$
         B) If $f(\mathbf{r})$ better than $f(\mathbf{x})$ then $\mathbf{x} = \mathbf{r}$ and fitness value
            $\mathrm{eval}(\mathbf{x}) = f(\mathbf{r})$
4) Return to step 2
```

performs significantly better than methods not designed for handling constraints like the existing DO algorithms. In stationary unconstrained group (fF, noC), also, as expected, the repair method in GA+Repair is no longer particularly useful. Fig. 2 shows that GA+Repair performs worse than all other methods in dynamic, unconstrained problems (dF, noC).

In the groups of *DCOPs* (fF+dC, dF+fC, dF+dC), things are different. As can be seen in Fig. 2, in DCOPs the difference between GA+Repair and GA is no longer as significant as it is in the stationary constrained case, meaning that the performance of GA+Repair significantly decreases. This happens in all three cases of DCOPs where only the constraints are dynamic (fF, dC), where only the objective functions are dynamic (dF, fC) and where both constraints and objective functions are dynamic (dF, dC).

Details of the impact of dynamic objective functions on the repair method can be seen in pair-wise comparisons in pairs 9 and 14 of Fig. 3 where GA+Repair is tested in pairs of almost identical constrained problems except that one has a fixed and the other has a dynamic objective function. As

[8]It is more difficult to analyze the effect of the repair strategy in the original Genocop III because this algorithm implements multiple CH strategies (beside the repair operator, there are ten other specialized operators to handle linear constraints).

[9]It should be noted that while Genocop III allows 25% of the repaired individuals to replace individuals in the population (Lamarckian evolution), in GA+Repair none of the repaired individuals is used to replace the original individuals (Baldwinian evolution). The reason is that in [31] it was found that Lamarckian evolution does not significantly increase/decrease the performance of Genocop III in solving DCOPs.

can be seen in these plots, the performance of GA+Repair significantly decreased in case the objective function is dynamic. The difference in performance of GA+Repair between the two problems of each pair is significantly larger than that of GA and existing DO algorithms, meaning that the presence of dynamic objective functions has a much greater impact on the repair method than on GA and existing DO methods.

Details of the impact of dynamic constraints on the repair method can be seen in the pair-wise comparisons in plot i of Fig. 3 and plot a of Fig. 4 (pairs of almost identical problems except that one has fixed and the other has dynamic constraints). Similar to the previous case, the results also show that the performance of GA+Repair is significantly decreased in case the constraints are dynamic and the presence of dynamic constraints has a much greater impact on the repair method than on existing DO methods.

However, although the presence of environmental dynamics does significantly decrease the performance of GA+Repair, in Fig. 2 it is interesting to see that the algorithm still performs better than existing DO algorithms in DCOPs (only that the difference becomes significantly smaller compared to the static constrained case). This shows the repair method has some characteristics that make it promising for solving DCOPs.

Another interesting, and somewhat counter-intuitive observation in our experiment is the presence of constraints does not make the problems more difficult to solve by GA+Repair. Instead, the presence of constraints always helps GA+Repair work better. Evidence can be found in the pair-wise comparison in pairs 1, 5, 6, 11, 12, 13 of Fig. 3 and in pair 21 of Fig. 4 where GA+Repair always performs better on the problem with constraints than on the problem without constraints. The experiment also shows that GA+Repair performs better where there is an infeasible barrier separating two feasible regions. Moreover, the larger the barrier, the better the performance of GA+Repair (see pairs 17, 18 in Fig. 4).

The experimental results confirm dynamics do have a significant effect on the performance of the repair method. Following is a further analysis to investigate if this effect is indeed caused by the outdated problem information (reference individuals and search individuals) and by the outdated strategy as suspected by our hypothesis.

b) *Analyze the behaviors of outdated reference individuals:* As mentioned earlier, outdated information might be caused by reference individuals having their objective values changed or even become infeasible after a change. The previously proposed measure *feasible reference individuals* was used to test if the algorithm is able to update the reference individuals properly. If the algorithm is able to update the reference individuals properly, it should be able to maintain a reference population of all feasible individuals during the search process.

The most suitable environments to test this behavior of the repair method are DCOPs with dynamic constraints where after each change the previous best feasible solutions are hidden by the moving infeasible region. They are G24_4, G24_5 (dF, dC) and G24_7 (fF, dC). As discussed earlier (see Fig. 2), in both groups the performance of GA+Repair decreases significantly compared to the case where the constraints are fixed (fF, fC). In these problems, if one or more reference individuals do become infeasible, there should be a drop in the total number of feasible reference individuals and this is one of the reasons making the repair method less effective.

The plot of feasible reference individuals of GA+Repair is given in Fig. 5. The figure shows, in all cases the original repair method was not able to keep all reference individuals feasible during the search. The number of feasible reference individuals drops to a very low level when a change occurs and most of the time the number of feasible reference individuals is much lower than five.

The results confirm the hypothesis that after a change, the population of reference individuals has become outdated due to the moving infeasible regions.

c) *Analyze the behaviors of the outdated balancing strategy:* In Section V-D2, it was suspected that individuals being outdated can also have a negative impact on the balancing strategy, which balances feasibility and infeasibility of the repair method. To test if the algorithm is still able to balance feasibility/infeasibility properly in dynamic environments, the proposed measure, feasible individuals in each disconnected region, was used to monitor the number of feasible individuals in each disconnected feasible region and the ratio of feasibility/infeasibility. The balancing mechanism should be able to manage a good distribution of individuals so that the better feasible regions should have more feasible individuals if it works well in the DCOP case.

The most suitable environments to test this behavior are DCOPs with two disconnected feasible regions where the global optimum keeps switching from one region to another after each change or after some consecutive changes. They are G24_1, G24_2, G24_3b, G24_4, G24_5, G24_6a, G24_6c, G24_6d, and G24_8b. All these problems belong to the group *SwO* in Fig. 2, where the performance of existing CH algorithms significantly decreases compared to the stationary constrained case (fF, fC). In such SwO problems, if the balancing mechanisms work well, at each change period the algorithm should be able to focus most feasible individuals on the region where the global optimum is currently in while still maintaining the same ratio of feasibility/infeasibility for diversity purposes.

The plot of feasible individuals in each disconnected region for GA+Repair is given in Fig. 6. The figure shows that in all cases except G24_3b, the repair method was not able to focus most feasible individuals on the region where the global optimum is currently in. Instead, the majority of feasible individuals still remained in one single region (region 2). The number of individuals in the other region (region 1) remained low regardless of where the global optimum is. This is due to the fact that, although the global optimum has switched to region 1, many individuals in region 2 were not updated and still have the outdated fitness values which might be even higher than the new global optimum value. These outdated individuals incorrectly attract a large number of individuals to the old feasible region. These results show that, due to its

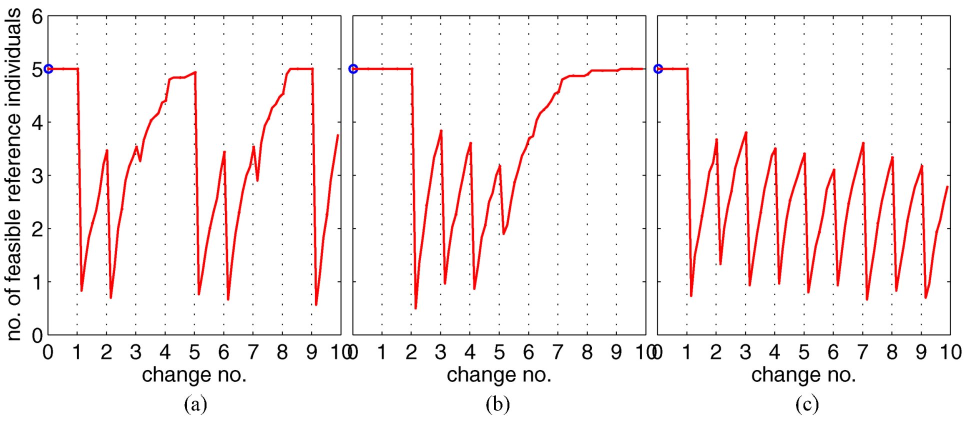

Fig. 5. Figure shows how GA+Repair maintains feasible reference individuals in problems with moving infeasible regions. The total number of reference individuals is five. The plot in the figures shows, among these five reference individuals, how many are actually feasible during the search process. (a) G24-4. (b) G24-5. (c) G24-7.

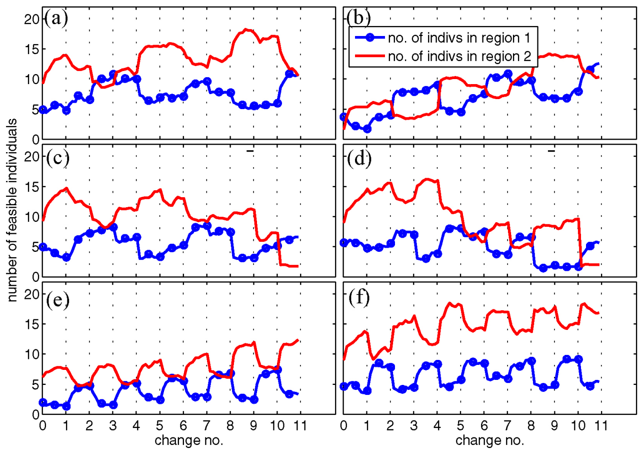

Fig. 6. This figure shows how the balance strategy of GA+Repair distributes its feasible individuals in disconnected feasible regions. The problems tested in this figure are those with global optima switching between two disconnected feasible regions. (a) GA+Repair in G24-1. (b) GA+Repair in G24-3b. (c) GA+Repair in G24-4. (d) GA+Repair in G24-5. (e) GA+Repair in G24-6a. (f) GA+Repair in G24-8b.

outdated strategy, the algorithm was not able to follow the switching optimum well.[10]

VI. Conclusion and Future Research

In this paper, we identified some special and not well-studied characteristics of DCOPs that might cause significant challenges to existing DO and CH strategies. Although these characteristics are common in real-world applications, in the continuous domain they have not been considered in most existing DO studies and they have not been captured in existing continuous DO benchmark problems.

A set of dynamic constrained benchmark problems for simulating the characteristics of DCOPs have been proposed to help close this gap. To help researchers assess algorithm performance in DCOPs, seven new measures have been proposed to evaluate the performance/analyse the behaviors of algorithms on dynamic unconstrained/constrained problems and one existing measure has also been modified to make it usable in DCOPs.

Using the newly proposed benchmark problems and measures, some literature reviews and detailed experimental analyses have been carried out to investigate the strengths and weaknesses of existing DO strategies (GA/RIGA/HyperM) and CH strategies (repair methods) in solving DCOPs. The experimental analyses reveal some interesting findings about the ability of existing algorithms in solving DCOPs. These findings can be categorized as follows.

First, three interesting findings about the performance of existing DO strategies in DCOPs have been identified: 1) the use of elitism might have a positive impact on the performance of existing diversity-maintaining strategies and but might have a negative impact on the performance of diversity-introducing strategies if they are not used with diversity-maintaining strategies; 2) the presence of infeasible areas has a negative impact on the performance of diversity-introducing/maintaining strategies; and 3) the presence of switching optima (between disconnected regions) has a negative impact on the performance of DO strategies if they are combined with penalty functions.

Second, it has been found that even if CH strategies can be combined with DO strategies, there might be two types of difficulties in applying existing CH strategies to solving DCOPs: 1) difficulties in handling dynamics, particularly in maintaining diversity and detecting changes; and 2) difficulties in handling constraints, which are caused by outdated CH strategies and problem-knowledge.

Third, some counter-intuitive behaviors were observed: the presence of constraints and dynamics in DCOPs might not always make the problems harder to solve. For example, the presence of constraints helps algorithms using the repair method like GA+Repair work better in the tested problems.

Finally, based on the findings about the strengths and weaknesses of some existing DO and CH strategies, a list

[10]Note that in the G24 set, individuals being outdated might not always be totally harmful because the changes in many problems are cyclic and hence the outdated individuals might actually play the role of memory elements to recall the previous good solutions. However, it is not clear how beneficial such memory elements could be, because the experiments show that GA+Repair still becomes less effective in the presence of environmental dynamics.

of possible requirements that DO and CH algorithms should meet to solve DCOPs effectively have been suggested. This list of requirements can be used as a guideline to design new algorithms to solve DCOPs in future research.

The results in this paper raise some open questions for future research. One direction is to develop new algorithms specialized in solving DCOPs based on our suggested list of requirements. Another direction is to investigate the impact of DCOPs' characteristics on other state-of-the-art CH and DO strategies. We are interested in investigating the performance of other adaptive feasibility/infeasibility balancing strategies, e.g., [38], [53] in DCOPs. We also plan to study the situations where the presence of constraints and dynamics would make it easier for certain classes of algorithms to solve DCOPs.

The research has some limitations to be improved in future research: memory-based approaches have not been considered in our analysis; the algorithms and methods used to analyze representative DO and CH strategies are very basic to keep the analysis at a manageable level; the analysis has been tested only in benchmark problems with unimodal objective functions; the types of changes are limited to linear and sinuous/cyclic changes and there was no consideration of hard/soft constraints. It would be interesting to extend the analysis on the multimodal set of benchmark problems in [18] and apply other types of changes such as random, chaotic, and nonlinear changes.

Acknowledgment

The authors would like to thank Prof. J. Branke, Prof. Y. Jin, Dr. S. Yang, C. Li, Dr. P. Rohlfshagen, Dr. T. Ray, Dr. J. Rowe, and Dr. A. Kaban for their fruitful discussions. They would also like to thank J. Rees, A. Economides, H. Maxwell, and H. Nguyen for their help in academic writing. The programs in this paper were developed from the source code provided by K. Williams [54] and Z. Michalewicz [33].

References

[1] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments: A survey," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 303–317, Jun. 2005.

[2] M. Andrews and A. L. Tuson, "Dynamic optimization: A practitioner requirements study," in *Proc. 24th Annu. Workshop U.K. Planning Scheduling Special Interest Group*, 2005.

[3] Y. Wang and M. Wineberg, "Estimation of evolvability genetic algorithm and dynamic environments," *Genet. Programming Evolvable Mach.*, vol. 7, no. 4, pp. 355–382, 2006.

[4] D. M. Prata, E. L. Lima, and J. C. Pinto, "Simultaneous data reconciliation and parameter estimation in bulk polypropylene polymerizations in real time," *Macromolecular Symposia*, vol. 243, no. 1, pp. 91–103, 2006.

[5] L. Araujo and J. J. Merelo, "A genetic algorithm for dynamic modeling and prediction of activity in document streams," in *Proc. 9th Annu. Conf. Genet. Evol. Comput.*, 2007, pp. 1896–1903.

[6] P. Tawdross, S. K. Lakshmanan, and A. Konig, "Intrinsic evolution of predictable behavior evolvable hardware in dynamic environment," in *Proc. 6th Int. Conf. Hybrid Intell. Syst.*, 2006, p. 60.

[7] M. Rocha, J. Neves, and A. Veloso, "Evolutionary algorithms for static and dynamic optimization of fed-batch fermentation processes," in *Adaptive and Natural Computing Algorithms*, B. Ribeiro, R. F. Albrecht, A. Dobnikar, D. W. Pearson, and N. C. Steele, Eds. Berlin, Germany: Springer, 2005, pp. 288–291.

[8] K. Deb, U. B. Rao, and S. Karthik, "Dynamic multiobjective optimization and decision-making using modified NSGA-II: A case study on hydro-thermal power scheduling," in *Proc. 4th Int. Conf. EMO*, LNCS 4403. Mar. 2007, pp. 803–817.

[9] P. Ioannou, A. Chassiakos, H. Jula, and R. Unglaub, "Dynamic optimization of cargo movement by trucks in metropolitan areas with adjacent ports," METRANS Transportation Center, Univ. Southern California, Los Angeles, CA, Tech. Rep., 2002 [Online]. Available: www.metrans.org/research/final/00-15_Final.htm

[10] K. Mertens, T. Holvoet, and Y. Berbers, "The DynCOAA algorithm for dynamic constraint optimization problems," in *Proc. 5th Int. Joint Conf. AAMAS*, 2006, pp. 1421–1423.

[11] J. M. Thompson and K. A. Dowsland, "A robust simulated annealing based examination timetabling system," *Comput. Oper. Res.*, vol. 25, nos. 7–8, pp. 637–648, 1998.

[12] U. Aickelin and K. Dowsland, "Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem," *J. Scheduling*, vol. 3, no. 3, pp. 139–153, 2000.

[13] H. Kim, "Target exploration for disconnected feasible regions in enterprise-driven multilevel product design," *Am. Instit. Aeronautics Astronautics J.*, vol. 44, no. 1, pp. 67–77, 2006.

[14] C. A. Liu, "New dynamic constrained optimization PSO algorithm," in *Proc. 4th ICNC*, 2008, pp. 650–653.

[15] H. Richter, "Memory design for constrained dynamic optimization problems," in *Proc. Eur. Conf. Applicat. Evol. Computat.*, LNCS 6024. 2010, pp. 552–561.

[16] T. T. Nguyen. (2011, Jan.). "Continuous dynamic optimization using evolutionary algorithms," Ph.D. thesis, School Comput. Sci., Univ. Birmingham, Birmingham, U.K. [Online]. Available: http://etheses.bham.ac.uk/1296, http://www.staff.ljmu.ac.uk/enrtngu1/theses/phd_thesis_nguyen.pdf

[17] T. T. Nguyen and X. Yao, "Benchmarking and solving dynamic constrained problems," in *Proc. IEEE CEC*, May 2009, pp. 690–697.

[18] T. T. Nguyen. (2008). "A proposed real-valued dynamic constrained benchmark set," School Comput. Sci., Univ. Birmingham, Birmingham, U.K., Tech. Rep. [Online]. Available: http://www.staff.livjm.ac.uk/enrtngu1/Papers/DCOP_benchmark.pdf

[19] T. T. Nguyen and X. Yao, "Dynamic time-linkage problem revisited," in *Proc. Eur. Workshops Applicat. Evol. Comput.*, LNCS 5484. 2009, pp. 735–744.

[20] C. Floudas, P. Pardalos, C. Adjiman, W. Esposito, Z. Gumus, S. Harding, J. Klepeis, C. Meyer, and C. Schweiger, *Handbook of Test Problems in Local and Global Optimization* (Nonconvex Optimization and Its Applications, vol. 33). Dordrecht, The Netherlands: Kluwer, 1999.

[21] H. G. Cobb, "An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments," Naval Res. Lab., Washington D.C., Tech. Rep. AIC-90-001, 1990.

[22] D. Parrott and X. Li, "Locating and tracking multiple dynamic optima by a particle swarm model using speciation," *IEEE Trans. Evol. Comput.*, vol. 10, no. 4, pp. 440–458, Aug. 2006.

[23] I. Moser and T. Hendtlass, "A simple and efficient multi-component algorithm for solving dynamic function optimization problems," in *Proc. IEEE CEC*, Sep. 2007, pp. 252–259.

[24] J. J. Grefenstette, "Genetic algorithms for changing environments," in *Proc. 2nd Parallel Problem Solving from Nature*, 1992, pp. 137–144.

[25] S. Yang and X. Yao, "Experimental study on population-based incremental learning algorithms for dynamic optimization problems," *Soft Comput. A Fusion Found. Methodol. Applicat.*, vol. 9, no. 11, pp. 815–834, 2005.

[26] T. Blackwell and J. Branke, "Multiswarms, exclusion, and anti-convergence in dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 10, no. 4, pp. 459–472, Aug. 2006.

[27] J. Branke, *Evolutionary Optimization in Dynamic Environments*. Dordrecht, The Netherlands: Kluwer, 2001.

[28] H. Richter, "Detecting change in dynamic fitness landscapes," in *Proc. IEEE CEC*, May 2009, pp. 1613–1620.

[29] H. Richter and S. Yang, "Learning behavior in abstract memory schemes for dynamic optimization problems," *Soft Comput.*, vol. 13, no. 12, pp. 1163–1173, 2009.

[30] D. Ayvaz, H. Topcuoglu, and F. Gurgen, "A comparative study of evolutionary optimization techniques in dynamic environments," in *Proc. 8th Annu. Conf. GECCO*, 2006, pp. 1397–1398.

[31] T. T. Nguyen and X. Yao, "Solving dynamic constrained optimization problems using repair methods," *IEEE Trans. Evol. Comput.*, 2010, submitted for publication [Online]. Available: http://www.staff.livjm.ac.uk/enrtngu1/Papers/Nguyen_Yao_dRepairGA.pdf

[32] K. A. Morales and C. Quezada, "A universal eclectic genetic algorithm for constrained optimization," in *Proc. 6th Eur. Congr. Intell. Soft Comput.*, 1998, pp. 518–522.
[33] Z. Michalewicz. (2009, Feb.). "The second version of Genocop III: A system which handles also nonlinear constraints," School Comput. Sci., Univ. Adelaide, Adelaide, Australia [Online]. Available: http://www.cs.adelaide.edu.au/~zbyszek/EvolSyst/gcopIII10.tar.Z
[34] T. T. Nguyen and X. Yao. (2010). "Detailed experimental results of GA, RIGA, HyperM and GA+Repair on the G24 set of benchmark problems," School Comput. Sci., Univ. Birmingham, Birmingham, U.K., Tech. Rep. [Online]. Available: http://www.staff.livjm.ac.uk/enrtngu1/Papers/DCOP_fulldata.pdf
[35] H. Motulsky, *Intuitive Biostatistics*, 1st ed. Oxford, U.K.: Oxford Univ. Press, Oct. 1995.
[36] T. Back, D. B. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation*. Bristol, U.K.: IOP Publishing Ltd., 1997.
[37] C. A. Coello Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art," *Comput. Methods Applied Mech. Eng.*, vol. 191, nos. 11–12, pp. 1245–1287, Jan. 2002.
[38] T. P. Runarsson and X. Yao, "Stochastic ranking for constrained evolutionary optimization," *IEEE Trans. Evol. Comput.*, vol. 4, no. 3, pp. 284–294, Sep. 2000.
[39] E. Mezura-Montes and C. A. Coello Coello, "A simple multimembered evolution strategy to solve constrained optimization problems," *IEEE Trans. Evol. Comput.*, vol. 9, no. 1, pp. 1–17, Feb. 2005.
[40] S. Venkatraman and G. G. Yen, "A generic framework for constrained optimization using genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 9, no. 4, pp. 424–435, Aug. 2005.
[41] T. P. Runarsson and X. Yao, "Search biases in constrained evolutionary optimization," *IEEE Trans. Syst. Man Cybern. Part C*, vol. 35, no. 2, pp. 233–243, May 2005.
[42] J. Joines and C. Houck, "On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with gas," in *Proc. 1st IEEE Conf. Evol. Comput.*, Jun. 1994, pp. 579–584.
[43] A. B. Hadj-Alouane and J. C. Bean, "A genetic algorithm for the multiple-choice integer program," *Oper. Res.*, vol. 45, no. 1, pp. 92–101, 1997.
[44] S. B. Hamida and A. Petrowski, "The need for improving the exploration operators for constrained optimization problems," in *Proc. CEC*, vol. 2. 2000, pp. 1176–1183.
[45] Z. Michalewicz and G. Nazhiyath, "Genocop III: A co-evolutionary algorithm for numerical optimization with nonlinear constraints," in *Proc. 2nd IEEE Int. Conf. Evol. Comput.*, Nov.–Dec. 1995, pp. 647–651.
[46] J. J. Grefenstette, R. Gopal, B. Rosmaita, and D. van Gucht, "Genetic algorithm for the TSP," in *Proc. 1st Int. Conf. Genet. Algorithms*, 1985, pp. 160–168.
[47] Z. Michalewicz, "Constraint-handling techniques: Decoders," in *Handbook of Evolutionary Computation*. Oxford, U.K.: Oxford Univ. Press, 1997, pp. C5.3:1–C5.3:3.
[48] G. Syswerda, "Schedule optimization using genetic algorithms," in *Handbook of Genetic Algorithms*, L. Davis, Ed. New York: Van Nostrand Reinhold, 1991, pp. 332–349.
[49] S. Salcedo-Sanz, "A survey of repair methods used as constraint handling techniques in evolutionary algorithms," *Comput. Sci. Rev.*, vol. 3, no. 3, pp. 175–192, 2009.
[50] R. Farmani and J. A. Wright, "Self-adaptive fitness formulation for constrained optimization," *IEEE Trans. Evol. Comput.*, vol. 7, no. 5, pp. 445–455, Oct. 2003.
[51] S. B. Hamida and M. Schoenauer, "ASCHEA: New results using adaptive segregational constraint handling," in *Proc. CEC*, vol. 1. 2002, pp. 884–889.
[52] T. Takahama and S. Sakai, "Constrained optimization by applying the alpha: Constrained method to the nonlinear simplex method with mutations," *IEEE Trans. Evol. Comput.*, vol. 9, no. 5, pp. 437–451, Oct. 2005.
[53] Y. Wang, Z. Cai, Y. Zhou, and W. Zeng, "An adaptive tradeoff model for constrained evolutionary optimization," *IEEE Trans. Evol. Comput.*, vol. 12, no. 1, pp. 80–92, Feb. 2008.
[54] K. P. Williams. (2008, Dec.). *Simple Genetic Algorithm (SGA) Source Code (in C)* [Online]. Available: http://www.kenwilliams.org.uk/code/ga2.c

Trung Thanh Nguyen (M'09) received the B.Sc. degree in computer science from Vietnam National University, Hanoi, Vietnam, in 2000, and the M.Phil. and Ph.D. degrees in computer science from the University of Birmingham, Birmingham, U.K., in 2007 and 2011, respectively.

From 2000 to 2005, he was a Researcher with the Research Institute of Post and Telecoms, Hanoi. In 2011, he was a Research Fellow with the University of Birmingham. He is currently a Research Fellow with the School of Engineering, Technology and Maritime Operations, Liverpool John Moores University, Liverpool, U.K. His current research interests include global (dynamic and static) optimization using metaheuristics and bioinspired methods, and applications of metaheuristics and operation research to real-world problems, especially problems in port/maritime environments.

Xin Yao (M'91–SM'96–F'03) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, China, in 1982, the M.Sc. degree from the North China Institute of Computing Technology, Beijing, China, in 1985, and the Ph.D. degree from USTC in 1990.

He was an Associate Lecturer and Lecturer with USTC from 1985 to 1990, a Post-Doctoral Fellow with Australian National University, Canberra, Australia, and with the Commonwealth Scientific and Industrial Research Organization, Melbourne, Australia, from 1990 to 1992, and a Lecturer, Senior Lecturer and Associate Professor with the University of New South Wales, Australian Defense Force Academy, Canberra, from 1992 to 1999. Since April 1999, he has been a Professor (Chair) of computer science with the School of Computer Science, University of Birmingham, Birmingham, U.K., where he is currently the Director of the Center of Excellence for Research in Computational Intelligence and Applications. He is also a Distinguished Visiting Professor (Grand Master Professorship) with USTC. He has more than 400 refereed publications. His major research interests include evolutionary computation and neural network ensembles.

Dr. Yao was a recipient of the 2001 IEEE Donald G. Fink Prize Paper Award, the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding 2008 Paper Award, the IEEE TRANSACTIONS ON NEURAL NETWORKS OUTSTANDING 2009 Paper Award, and several other Best Paper Awards. He is a Distinguished Lecturer of the IEEE Computational Intelligence Society, an Invited Keynote/Plenary Speaker of 70 international conferences, and a former Editor-in-Chief of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION from 2003 to 2008.