Chapter 8
Evolutionary Optimization on Continuous Dynamic Constrained Problems – An Analysis

Trung Thanh Nguyen and Xin Yao

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 little research on whether current algorithms work well on continuous dynamic constrained optimization problems (DCOPs). This chapter investigates this issue. The chapter will present some studies on the characteristics that can make DCOPs difficult to solve by some existing dynamic optimization (DO) algorithms. We will then introduce a set of benchmark problems with these characteristics and test several representative DO strategies on these problems. The results confirm that DCOPs do have special characteristics that can significantly affect algorithm performance. Based on the analyses of the results, a list of potential requirements that an algorithm should meet to solve DCOPs effectively will be proposed.

8.1 Introduction

This chapter attempts to investigate the characteristics, difficulty and solutions of a very common class of problem - dynamic constrained optimization problems (DCOPs). DCOPs are constrained optimization problems that have two properties: (a) the objective functions, the constraints, or both, may change over time, and (b) the changes are taken into account in the optimization process. It is believed that a majority of real-world dynamic problems are DCOPs. However, there are few

Trung Thanh Nguyen
School of Engineering, Technology and Maritime Operations,
Liverpool John Moores University, Liverpool L3 3AF, U.K.
e-mail: `T.T.Nguyen@ljmu.ac.uk`

Xin Yao
Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, University of Birmingham,
Birmingham B15 2TT, U.K.
e-mail: `X.Yao@cs.bham.ac.uk`

DOI: 10.1007/978-3-642-38416-5_8

studies on continuous dynamic constrained optimization. Existing studies in continuous dynamic optimization only focus on the unconstrained or domain constraint dynamic cases (which in this chapter both are regarded as "unconstrained" problems).

This lack of attention to 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 strategies perform in dynamic constrained environments if most of them are designed for and tested in unconstrained dynamic 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 DCOPs, finding the answers to the questions above is essential to have better understanding about the practical issues of DCOPs and to solve this class of problem more effectively. Note that this chapter only investigates the impact of DCOPs on dynamic optimization strategies. For a study on the impact of DCOPs on constraint handling strategies, readers are referred to [23].

The chapter is organized as follows. Section 8.2 discusses the special characteristics from real-world DCOPs and discuss how the characteristics make DCOPs different from unconstrained dynamic optimization problems (DOPs). Section 8.3 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. Section 8.4 discusses the possibility of solving DCOPs using some representative DO strategies. Experimental analyses about the strengths and weaknesses, and the effect of the mentioned characteristics on each strategy will be reported. Based on the experimental results, a list of requirements that algorithms should meet to solve DCOPs effectively is proposed. Finally, Section 8.5 concludes the chapter and identifies future directions.

8.2 Characteristics of Real-World Dynamic Constrained Problems

Constraints make real-world DCOPs 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: (a) both the objective function and the constraints are dynamic [2, 27, 35]; (b) only the objective function is dynamic while the constraints are static [3, 31, 33]; and (c) the objective function is static and the constraints are dynamic [8, 12, 15]. In all three types, the presence of infeasible areas can affect how the global optimum moves, or appears each change. This leads to some special characteristics which are not found in the unconstrained cases 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 [1, 13, 34]. 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.

8.3 A Real-Valued Benchmark to Simulate DCOPs Characteristics

8.3.1 Related Literature

In the continuous domain, there is no existing continuous benchmark that fully reflects the characteristics of DCOPs listed in Section 8.2. Among existing continuous benchmarks, there are only few 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 properites of DCOPs.

The second study was [29]. 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 Fig. 2 in [29]). Although the framework used to generate this benchmark problem is highly configurable, the current single benchmark problem generated by the framework in [29] was designed for a different purpose and hence does not simulate the properties mentioned in Section 8.2. 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 third study was [39]. Based on an existing static test problem [9], the authors assigned six pre-defined values (scalars or matrices) to the coefficients of this static functions to represent six different time steps. Because only six values were given to each coefficient, the dynamic of the problems were defined only up to six time steps. This prevents users from testing the problem in the long run. In addition, there appears to be no specified rule for the dynamics, making it difficult to simulate any dynamic rules from real-world applications. Besides this limitation, the problem also does not reflect common situations such as dynamic objective + fixed constraints, fixed objective + dynamic constraints. It is also unclear if other common properites of DCOPs can be simulated.

The lack of benchmark problems for DCOPs makes it difficult to (a) evaluate how well existing DO algorithms would work on DCOPs, and (b) design new algorithms specialising in DCOPs. Given that a majority of recent real-world DOPs are DCOPs [20], 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 [21]. This framework then was extended in [23] 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, were developed. In this chapter we will describe the benchmark set with unimodal objective functions (many problems in the set still have multiple optima due to the constraints) in detail. Detailed descriptions of the multimodal, scalable set can be found in a technical report [19].

8.3.2 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 generalise $f_P(x)$ 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 [19]. Some additional information is provided in [22] (Section 3).

8.3.3 A Dynamic Constrained Benchmark Set

A set of 18 benchmark problems named G24[1] was introduced using the new procedure described in the previous subsection. The general form for each problem in the G24 set is as follows:

[1] This benchmark set was named after a static function originally from in [10]. This static function was named G24 in the "CEC06 competition on constrained real-parameter optimization". The static G24 function was adapted to create the $f^{(1)}$, $g^{(1)}$and $g^{(2)}$ function forms of this DCOP G24 benchmark set.

Table 8.1 The objective function form and set of constraint function forms for each problem

Benchmark problem	objective function
G24_8a & G24_8b	$f(x) = f^{(2)}$
All other problems	$f(x) = f^{(1)}$
Benchmark problem	**Set G of constraints**
G24_u; G24_uf; G24_2u; G24_8a	$G = \{\emptyset\}$
G24_6a	$G = \left\{g^{(3)}, g^{(6)}\right\}$
G24_6b	$G = \left\{g^{(3)}\right\}$
G24_6c	$G = \left\{g^{(3)}, g^{(4)}\right\}$
G24_6d	$G = \left\{g^{(5)}, g^{(6)}\right\}$
All other problems	$G = \left\{g^{(1)}, g^{(2)}\right\}$

$$\begin{aligned} &\text{minimise } f(\mathbf{x}) \\ &\text{subject to } g_i(\mathbf{x}) \leq 0,\ g_i(\mathbf{x}) \in G,\ i = 1,..,n \end{aligned}$$

where the objective function $f(\mathbf{x})$ can be one of the function forms set out in Eq. (8.1-8.2), each constraint $g_i(\mathbf{x})$ can be one of the function forms given in Eq. (8.3-8.8), 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 Table 8.1.

Eqs. (8.1-8.2) describe the general function forms for the objective functions in the G24 set. Of these function forms, $f^{(2)}$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 [10] and $f^{(2)}$is a newly designed function.

$$f^{(1)} = -(X_1 + X_2) \tag{8.1}$$

$$f^{(2)} = -3\exp\left(-\sqrt{\sqrt{(X_1)^2 + (X_2)^2}}\right) \tag{8.2}$$

where $X_i = X_i(x_i, t) = p_i(t)(x_i + q_i(t))$;$0 \leq x_1 \leq 3$;$0 \leq x_2 \leq 4$ 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.

Eqs. (8.3-8.8) describe 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 [10] and $g^{(3)}$,$g^{(4)}$, $g^{(5)}$ and $g^{(6)}$ are newly designed functions.

$$g^{(1)} = -2Y_1^4 + 8Y_1^3 - 8Y_1^2 + Y_2 - 2 \tag{8.3}$$

$$g^{(2)} = -4Y_1^4 + 32Y_1^3 - 88Y_1^2 + 96Y_1 + Y_2 - 36 \tag{8.4}$$

$$g^{(3)} = 2Y_1 + 3Y_2 - 9 \tag{8.5}$$

$$g^{(4)} = \begin{cases} -1 \text{ if } (0 \le Y_1 \le 1)\text{or}(2 \le Y_1 \le 3) \\ 1 \text{ otherwise} \end{cases} \tag{8.6}$$

$$g^{(5)} = \begin{cases} -1 \text{ if } (0 \le Y_1 \le 0.5)\text{or}(2 \le Y_1 \le 2.5) \\ 1 \text{ otherwise} \end{cases} \tag{8.7}$$

$$g^{(6)} = \begin{cases} -1 \text{ if } [(0 \le Y_1 \le 1) \text{ and } (2 \le Y_2 \le 3)] \\ \quad \text{or } (2 \le Y_1 \le 3) \\ 1 \text{ otherwise} \end{cases} \tag{8.8}$$

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 Eqs. (8.3-8.8), 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 functions 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 8.2.

Two guidelines were used to design the test problems: (a) problems should simulate the common properties of DCOPs as mentioned in Section 8.2 and (b) 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 the two problems in the pair, it is possible to analyse 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 in [23] (Table 8.2). Each test problem is able to capture one or several of the mentioned characteristics of DCOPs, as shown in Table 8.3. In addition, the problems and their relationships are carefully designed so that they can be arranged in 21 pairs (Table 8.4), 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).

Table 8.2 Dynamic parameters for all test problems in the benchmark set G24. Each dynamic parameter is a time-dependant rule/function which governs the way the problems change (reproduced with permission from [23])

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

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

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	$p_i(t) = r_i(t) = 1;\ q_i(t) = s_1(t) = 0;$ $s_2(t) = t.\frac{x_2 \max - x_2 \min}{S}$
G24_8a	$p_i(t) = -1; q_1(t) = -(c_1 + r_a.\cos(k\pi t))$ $q_2(t) = -(c_2 + r_a.\sin(k\pi t));$
G24_8b	$p_i(t) = -1; q_1(t) = -(c_1 + r_a.\cos(k\pi t))$ $q_2(t) = -(c_2 + r_a.\sin(k\pi t)); r_i(t) = 1;\ s_i(t) = 0$
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, $i = 1, 2$

Table 8.3 Properties of each test problem in the G24 benchmark set (reproduced with permission from [23])

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&No	Yes&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&No	Yes&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
Yes&No	It means OICB/OISB is true at some other changes and false at some others
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

8.4 Challenges to Solve DCOPs

8.4.1 Analysing the Performance of Some Common Dynamic Optimization Strategies in Solving DCOPs

The purpose of this section is to discuss whether the DO strategies commonly used in existing literature can be applied directly to solving DCOPs. We also report our analyses in [23] of whether the special characteristics of DCOPs might have any effect on the performance of these strategies and why. The results of the analysis will also provide insight in understanding how to design suitable algorithms for solving DCOPs.

Table 8.4 The 21 test cases (pairs) to be used in this chapter (reproduced with permission from [23])

Static problems: Unconstrained vs Fixed constraints			
1	G24_uf (fF, noC)	vs	G24_f (fF, fC)
Fixed objectives vs Dynamic objectives			
2	G24_uf (fF, noC)	vs	G24_u (dF, noC)
3	G24_f (fF, fC, OICB)	vs	G24_1 (dF, fC, OICB)
4	G24_f (fF, fC, OICB)	vs	G24_2 (dF, fC, ONICB)
Dynamic objectives: Unconstrained vs Fixed constraints			
5	G24_u (dF, noC)	vs	G24_1 (dF, fC, OICB)
6	G24_2u (dF, noC)	vs	G24_2 (dF, fC, ONICB)
Fixed constraints vs Dynamic constraints			
7	G24_1 (dF, fC, OICB)	vs	G24_4 (dF, dC, OICB)
8	G24_2 (dF, fC, ONICB)	vs	G24_5 (dF, dC, ONICB)
9	G24_f (fF, fC)	vs	G24_7 (fF, dC, NNAO)
10	G24_3f (fF, fC)	vs	G24_3 (fF, dC, NAO)
No constraint vs Dynamic constraints			
11	G24_u (dF, noC)	vs	G24_4 (dF, dC, OICB)
12	G24_2u (dF, noC)	vs	G24_5 (dF, dC, ONICB)
13	G24_uf (fF, noC)	vs	G24_7 (fF, dC)
Moving constraints expose better optima vs not expose optima			
14	G24_3f (fF, fC)	vs	G24_3 (fF, dC, NAO)
15	G24_3 (fF, dC, NAO)	vs	G24_3b (dF, dC, NAO)
Connected feasible regions vs Disconnected feasible regions			
16	G24_6b (1R)	vs	G24_6a (2DR, hard)
17	G24_6b (1R)	vs	G24_6d (2DR, hard)
18	G24_6c (2DR, easy)	vs	G24_6d (2DR, hard)
Optima in constraint boundary vs Optima NOT in constr boundary			
19	G24_1 (dF, fC, OICB)	vs	G24_2 (dF, fC, ONICB)
20	G24_4 (dF, dC, OICB)	vs	G24_5 (dF, dC, ONICB)
21	G24_8b (dF, fC, OICB)	vs	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 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 (EA) 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-initialising/re-locating the individuals. This strategy was introduced years ago [7] but is still extensively used [18, 26].

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 [11]. This strategy is still the main strategy in many recent approaches [5, 38].

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 [7] 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 to achieve better performance. Typical examples are the multi-population/multi-swarm approaches, where multiple sub-populations are used to maintain diversity and each sub-population/sub-swarm focuses on tracking one single optimum [5, 6].

8.4.2 Chosen Algorithms and Experimental Settings

8.4.2.1 Chosen Algorithms

Two commonly used algorithms: *triggered hyper-mutation GA* (HyperM [7]) and *random-immigrant GA* (RIGA [11]) 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 analyse their behaviour. 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

Table 8.5 Test settings for all algorithms used in the chapter

All the algorithms (exceptions below)	Pop size (pop_size)	5, 15, 25 (medium), 50, 100
	Elitism	Elitism & non-elitism if applicable
	Selection method	Non-linear ranking as in [16]
	Mutation method	Uniform, $P = 0.15$.
	Crossover method	Arithmetic, $P = 0.1$.
HyperM	Triggered mutate	Uniform, $P = 0.5$ as in [7].
RIGA	Rand-immig. rate	$P = 0.3$ as in [11].
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)

mutation strategy), it would be easier to compare/analyse their performance. The performance of HyperM and RIGA was also compared with a basic GA to see if they work well on the tested problems.

8.4.2.2 Parameter Settings

Table 8.5 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 8.3. 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, 30]) and 0.2 ([4, 6]). For HyperM and RIGA, the best *hyper-mutation-rate* and *random-immigrant-rate* parameter values observed in the original papers [7, 11] were used. The same implementations as described in [7] and [11] were used to reproduce these two algorithms. A crossover rate of 0.1 was chosen for all algorithms because, according to the analysis in [25], 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 [25] where it was found that the overall behaviours of the algorithms are not different from those using the default/best known settings, except for the followings: (i) When the base mutation rate is very low (≤ 0.01), the performance of GA and HyperM drop significantly; (ii) 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. (iii) 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; (iv) The suitable range of crossover rate is 0.1-1.0.

8.4.2.3 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 [17] 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 parameter [17].

8.4.2.4 Performance Measures

For measuring the performance of the algorithms in this particular experiment, an existing measure: the *offline error* [6] was modified. The measure is calculated as the average over, at every evaluation, 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.

$$E_{MO} = \frac{1}{num\_of\_gen} \sum_{j=1}^{num\_of\_gen} e_{MO}(j) \tag{8.9}$$

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

Five new measures were also proposed to analyse 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 analyse the convergence behaviour of algorithms in dynamic environments. The RR measure is used to analyse *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.

$$RR = \frac{1}{m}\sum\nolimits_{i=1}^{m} \frac{\sum_{j=1}^{p(i)} \left[f_{best}(i,j) - f_{best}(i,1)\right]}{p(i)\left[f_{best}(i,p(i)) - f_{best}(i,1)\right]} \tag{8.10}$$

where $f_{best}(i,j)$ 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 [2].

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 *absolute recovery rate* (ARR). This measure is very similar to the RR but is used to analyse *how quick it is for an algorithm to start converging to the global optimum before the next change occurs*:

$$ARR = \frac{1}{m}\sum\nolimits_{i=1}^{m} \frac{\sum_{j=1}^{p(i)} \left[f_{best}(i,j) - f_{best}(i,1)\right]}{p(i)\left[f^{*}(i) - f_{best}(i,1)\right]} \tag{8.11}$$

where $f_{best}(i,j), i, j, m, p(i)$ are the same as in Eq. 8.10 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[3].

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 takes to converge. The *RR-ARR diagram* in Fig. 8.1 shows some analysis guidelines.

A third measure, *percentage of selected infeasible individuals*, is proposed to analyse 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 analyse the behaviour of algorithms using triggered-mutation mechanisms such as HyperM, a fourth measure: *triggered-time count*, which counts the number

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

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

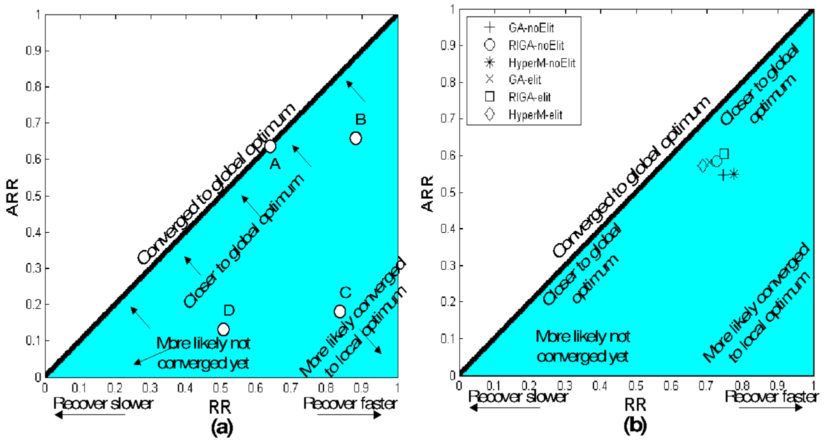

Fig. 8.1 Diagram (a) provides a guideline for analysing the convergence behaviour/recovery speed of an algorithm given its RR/ARR scores. These scores can be represented as the x and y coordinations of a point on the diagonal thick line or inside the shaded area. The position of the point represents the behaviour 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; and point B - recover quickly). **Diagram (b)** shows the mapping of the RR/ARR scores of GA, RIGA, and HyperM to the RR-ARR diagram. (Reproduced with permission from [23]).

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 v generations after a change, where v 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 all of the measures used here are specifically designed for dynamic problems. This creates a problem for the experiments in this chapter because in the G24 benchmark set there are not only dynamic problems but also stationary problems. To overcome this issue, in this study 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.

8.4.3 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 [24]. These data were further analysed 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 of all, the average performance of the tested algorithms on each major group of problems under different parameter settings and dynamic ranges were summarised to have an overall picture of algorithm behaviour on different types of problems (see Fig. 8.2). Then the effect of each problem characteristic on each algorithm was analysed 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 8.4 of Section 8.3 (see test results in Figs. 8.4 and 8.5). For 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 summarised results are presented in Fig. 8.2 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 [24].

The experimental results show some interesting findings.

8.4.3.1 The Impact of Different Dynamic Ranges on Algorithm Performance

The summarised results in groups of problems (Fig. 8.2) show that (i) generally the behaviour of algorithms and their relative strengths/weaknesses in comparison with other algorithms still remain roughly the same when the dynamic settings change; and (ii) as expected in most cases algorithms' performance decrease 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 and in population size.

8.4.3.2 The Effect of Elitism on Algorithm Performance

The summarised results in groups of problems (Fig. 8.2) and the pair-wise comparisons in Fig. 8.4 and Fig. 8.5 reveal an interesting effect of elitism on both unconstrained and constrained dynamic cases: the elitism versions of GA/RIGA/HyperM perform better than their non-elitism 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 8.4.2.4: *recovery rate* (RR) and *absolute recovery rate* (ARR) were used to study the inefficiency of GA/RIGA/HyperM in

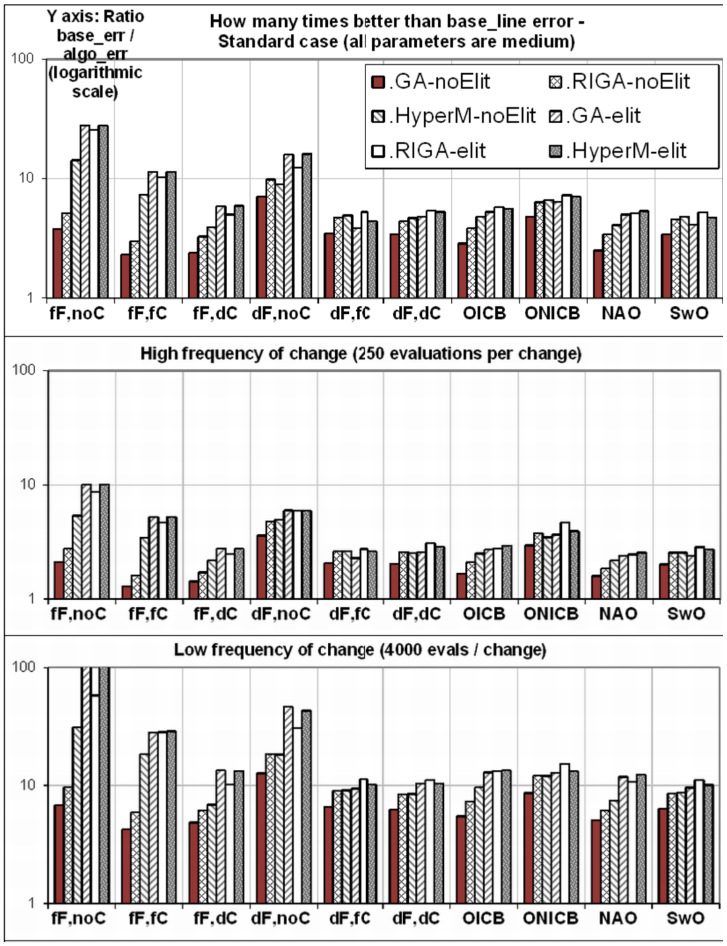

Fig. 8.2 Algorithm performance in groups of problem (part 1 - see Fig. 8.3 for part 2). Performance (vertical axis in logarithmic scale) is evaluated by calculating the ratio between the *base line* (worst error among all scenarios) and the error of each algorithm in each problem to see how many times their performance is better (smaller) than the *base line*. Explanations for abbreviations can be found in Table 8.4.

the non-elitism case. The scores of the algorithms on these measures are given in Fig. 8.1b. 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 non-elitism version is closer to the bottom-left corner, meaning that non-elitism makes RIGA converge slower/less accurately. Finally, for GA/HyperM, their elitism versions are closer to the global optimum while their non-elitism versions are closer to the bottom-right corner, meaning that the

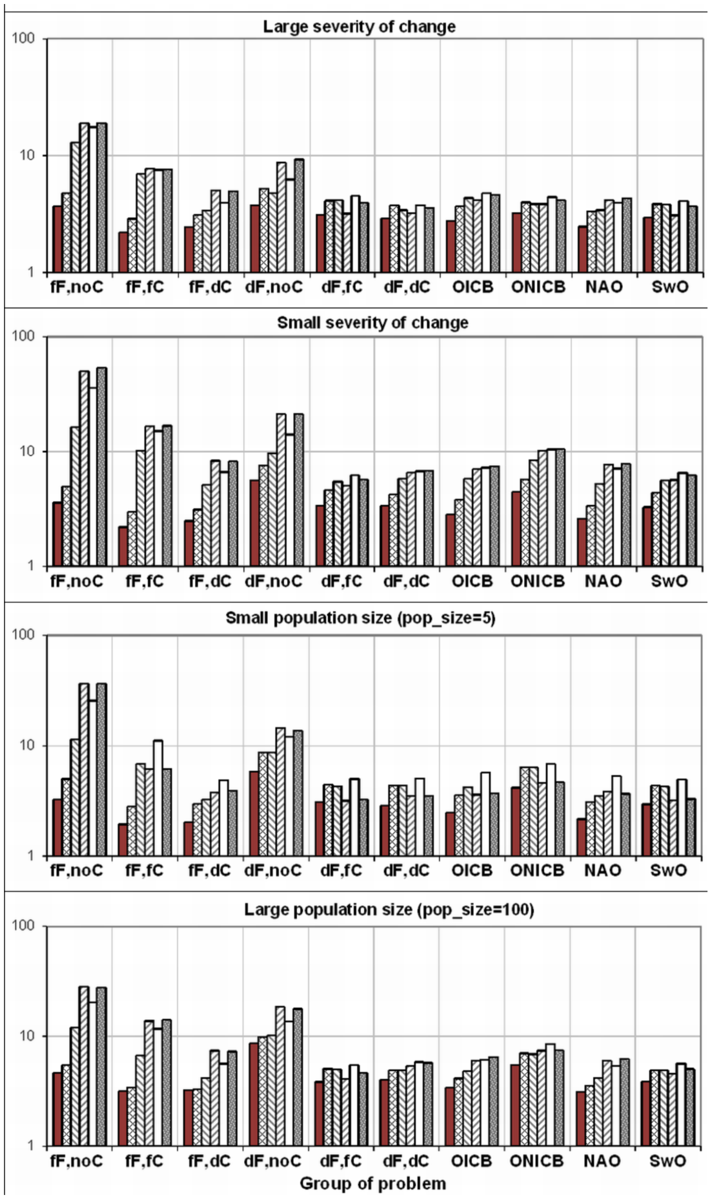

Fig. 8.3 Algorithm performance in groups of problem (part 2 - see Fig. 8.2 for part 1 and explanation)

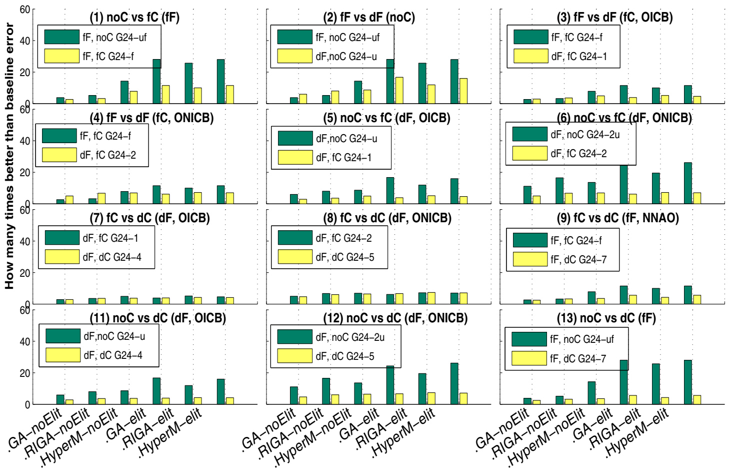

Fig. 8.4 The effect of twelve different problem characteristics on algorithm performance (medium case). Performance (vertical axis) is evaluated based on the ratio between the base line error (described in Figure 8.2) and algorithm errors. Each subplot represents algorithm performance (pair of adjacent 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 represent the test case numbers (in brackets) followed by an abbreviated description. Explanations for the abbreviations are in the last rows of Table 8.4.

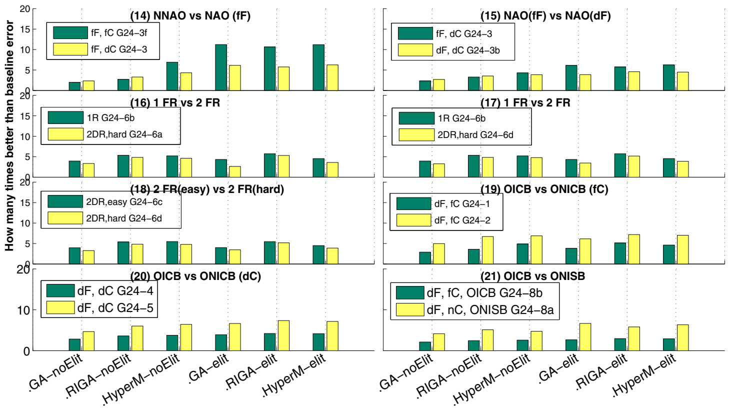

Fig. 8.5 The effect of the other eight different problem properties on algorithm performance (medium case). Instructions to read this figure can be found in Figure 8.4.

non-elitism versions of GA/HyperM are more suceptible 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 comes with a trade-off: the convergence speed is affected. In such a situation, elitism can be used to speed up the convergence process. Elite members can guide the population to exploit the good regions faster while still maintaining diversity.

8.4.3.3 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 behaviour can be seen in Fig. 8.2 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 behaviour can also be seen in the more accurate pair-wise comparisons in Fig. 8.4 and Fig. 8.5: 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. 8.4 and pair 21 in Fig. 8.5).

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 penalised with lower fitness values by the penalty functions. Consequently, these diversified individuals cannot be used for maintaining diversity 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 8.6, 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[4].

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.

[4] Non-elitism algorithms are able to retain more infeasible individuals, of which some might be diversified solutions. However, as shown in Subsection 8.4.3.2, in the non-elitism case this higher percentage of infeasible individuals comes with a trade-off of slower/less accurate convergence, which leads to the generally poorer performance.

Table 8.6 Average *percentage of selected infeasible individuals* over 18 problems. The last row shows the average *percentage of infeasible areas*. (Reproduced with permission from [23])

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%**

8.4.3.4 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 it is necessary 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 that 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 8.4 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. 8.5) 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.

Table 8.7 The *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. (Reproduced with permission from [23])

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

8.4.3.5 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 [21], 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 analyse how the triggered-hypermutation mechanism works on problem G24_3. As can be seen in Table 8.7, HyperM either was not able to trigger its hyper-mutation rate to deal with changes (elitism case, *triggered-time count*=0 & *detected-change count*=0) or was not able to trigger its hyper-mutation rate correctly when a change occurs (non-elitism case, *triggered-time count*∼188.7 & *detected-change count*∼1.74). It is worth noting in the non-elitism case, most of the trigger times are caused by the selection process because in non-elitism selection the best solution in the population is not always selected for the next generation.

Table 8.7 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.

8.4.4 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:

(i) 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.

(ii) 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 and search in both regions and infeasible regions.

(iii) 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.

Recent experimental results have shown that the directions above could be helpful for improving the performance of DO algorithms in solving DCOPs. The use of elitism was shown to have positive effects in [25, 37], detecting and/or searching in infeasible areas helped improve performance in [25, 29, 32], and tracking feasible areas gave superior results in [21, 25].

8.5 Conclusion and Future Research

In this chapter we have reviewed some important and not well studied characteristics of DCOPs that might cause significant challenges to existing DO 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 most existing continuous DO benchmark problems.

A set of dynamic constrained benchmark problems for simulating the characteristics of DCOPs, together with eight performance measures, have been discussed to help close this gap.

Using the benchmark problems and measures, we discussed detailed experimental analyses to investigate the strengths and weaknesses of existing DO strategies (GA/RIGA/HyperM) in solving DCOPs. The experimental analyses reveal some interesting findings about the ability of existing algorithms in solving DCOPs. These findings can be categorised as follows.

First, three findings about the performance of existing DO strategies in DCOPs have been identified: (a) the use of elitism might have a positive impact on the performance of existing diversity-maintaining strategies but might have a negative impact on the performance of diversity-introducing strategies if they are not used with diversity-maintaining strategies; (b) the presence of infeasible areas has a negative impact on the performance of diversity-introducing/maintaining strategies; and (c) 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, based on the findings about the strengths and weaknesses of some existing DO strategies, a list of possible requirements that DO 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 and discussions in this chapter raise some open questions for future research. One direction is to develop new algorithms specialised in solving DCOPs based on our suggested list of requirements. We also plan to apply the results achieved in this chapter to real-world applications, especially to dynamic environments such as container terminals where there is the need to provide dynamic optimization solutions for such problems as dynamic scheduling of automatic-guided vehicles, dynamic allocation of quay-side and stack-side cranes, and dynamic stacking of containers.

Acknowledgements. This work was partially supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/E058884/1, a UK ORS Award, a studentship from the School of Computer Science, University of Birmingham and an EU-funded project named ”Intelligent Transportation for Dynamic Environment (InTraDE)”. The programs in this chapter were developed from the source code provided by Williams [36].

References

[1] Aickelin, U., Dowsland, K.: Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. J. of Sched. 3, 139–153 (2000)

[2] Andrews, M., Tuson, A.L.: Dynamic optimisation: A practitioner requirements study. In: Proc. 24th Annual Workshop of the UK Planning and Scheduling Special Interest Group (2005)

[3] Araujo, L., Merelo, J.J.: A genetic algorithm for dynamic modelling and prediction of activity in document streams. In: Proc. 9th Annual Conf. Genetic and Evol. Comput., pp. 1896–1903 (2007)

[4] Ayvaz, D., Topcuoglu, H., Gurgen, F.: A comparative study of evolutionary optimization techniques in dynamic environments. In: Proc. 8th Annual Conf. Genetic and Evol. Comput., pp. 1397–1398 (2006)

[5] Blackwell, T., Branke, J.: Multiswarms, exclusion, and anti-convergence in dynamic environments. IEEE Trans. Evol. Comput. 10(4), 459–472 (2006)

[6] Branke, J.: Evolutionary Optimization in Dynamic Environments. Kluwer (2001)

[7] Cobb, H.G.: An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuouis, time-dependent nonstationary environments. Tech. Rep. AIC-90-001, Naval Research Laboratory, Washington, USA (1990)

[8] Deb, K., Rao N., U.B., Karthik, S.: Dynamic multi-objective optimization and decision-making using modified NSGA-II: A case study on hydro-thermal power scheduling. In: Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., Murata, T. (eds.) EMO 2007. LNCS, vol. 4403, pp. 803–817. Springer, Heidelberg (2007)

[9] Deb, K., Saha, A.: Finding multiple solutions for multimodal optimization problems using a multi-objective evolutionary approach. In: Proc. 12th Annual Conf. Genetic and Evol. Comput., pp. 447–454 (2010)

[10] Floudas, C., Pardalos, P., Adjiman, C., Esposito, W., Gumus, Z., Harding, S., Klepeis, J., Meyer, C., Schweiger, C.: Handbook of Test Problems in Local and Global Optimization. In: Noncovex Optimization and Its Applications, vol. 33. Kluwer Academic Publishers (1999)

[11] Grefenstette, J.J.: Genetic algorithms for changing environments. In: Proc. 2nd Int. Conf. Parallel Problem Solving from Nature, pp. 137–144 (1992)

[12] Ioannou, P., Chassiakos, A., Jula, H., Unglaub, R.: Dynamic optimization of cargo movement by trucks in metropolitan areas with adjacent ports. Tech. Rep., METRANS Transportation Center, University of Southern California, Los Angeles, CA 90089, USA (2002), `http://www.metrans.org/research/final/00-15_Final.htm`

[13] Kim, H.: Target exploration for disconnected feasible regions in enterprise-driven multilevel product design. American Institute of Aeronautics and Astronautics Journal 44(1), 67–77 (2006)

[14] Liu, C.A.: New dynamic constrained optimization pso algorithm. In: Proc. 4th Int. Conf. Natural Comput., pp. 650–653 (2008)

[15] Mertens, K., Holvoet, T., Berbers, Y.: The DynCOAA algorithm for dynamic constraint optimization problems. In: Proc. 5th Int. Joint Conf. Autonomous Agents and Multiagent Syst., pp. 1421–1423 (2006)

[16] Michalewicz, Z.: The second version of Genocop III: a system which handles also nonlinear constraints, `http://www.cs.adelaide.edu.au/zbyszek/EvolSyst/gcopIII10.tar.Z` (accessed February 2009)

[17] Morales, K.A., Quezada, C.: A universal eclectic genetic algorithm for constrained optimization. In: Proc. 6th Europ. Congr. Intell. & Soft Comput., pp. 518–522 (1998)

[18] Moser, I., Hendtlass, T.: A simple and efficient multi-component algorithm for solving dynamic function optimisation problems. In: Proc. 2007 IEEE Congr. Evol. Comput., pp. 252–259 (2007)

[19] Nguyen, T.T.: A proposed real-valued dynamic constrained benchmark set. Tech. Rep., School of Computer Science, Univesity of Birmingham (2008), `http://www.staff.ljmu.ac.uk/enrtngu1/Papers/DCOPbenchmark.pdf`

[20] Nguyen, T.T.: Continuous Dynamic Optimisation Using Evolutionary Algorithms. Ph.D. thesis, School of Computer Science, University of Birmingham (2011), `http://etheses.bham.ac.uk/1296` and `http://www.staff.ljmu.ac.uk/enrtngu1/theses/phdthesisnguyen.pdf`

[21] Nguyen, T.T., Yao, X.: Benchmarking and solving dynamic constrained problems. In: Proc. 2009 IEEE Congr. Evol. Comput., pp. 690–697 (2009)

[22] Nguyen, T.T., Yao, X.: Dynamic time-linkage problems revisited. In: Giacobini, M., et al. (eds.) EvoWorkshops 2009. LNCS, vol. 5484, pp. 735–744. Springer, Heidelberg (2009)

[23] Nguyen, T.T., Yao, X.: Continuous dynamic constrained optimisation - the challenges. IEEE Trans. Evol. Comput. 166, 769–786 (2012)

[24] Nguyen, T.T., Yao, X.: Detailed experimental results of GA, RIGA, HyperM and GA+Repair on the G24 set of benchmark problems. Tech. Rep., School of Computer Science, University of Birmingham (2010), `http://www.staff.ljmu.ac.uk/enrtngu1/Papers/DCOPfulldata.pdf`

[25] Nguyen, T.T., Yao, X.: Solving dynamic constrained optimisation problems using stochastic ranking and repair methods. IEEE Trans. Evol. Comput. (2010) (submitted), `http://www.staff.ljmu.ac.uk/enrtngu1/Papers/NguyenYaodRepairGA.pdf`

[26] Parrott, D., Li, X.: Locating and tracking multiple dynamic optima by a particle swarm model using speciation. IEEE Trans. Evol. Comput. 10(4), 440–458 (2006)

[27] Prata, D.M., Lima, E.L., Pinto, J.C.: Simultaneous data reconciliation and parameter estimation in bulk polypropylene polymerizations in real time. Macromolecular Symposia 243(1), 91–103 (2006)

[28] Richter, H.: Detecting change in dynamic fitness landscapes. In: Proc. 2009 IEEE Congr. Evol. Comput., pp. 1613–1620 (2009)

[29] Richter, H.: Memory design for constrained dynamic optimization problems. In: Di Chio, C., et al. (eds.) EvoApplicatons 2010, Part I. LNCS, vol. 6024, pp. 552–561. Springer, Heidelberg (2010)

[30] Richter, H., Yang, S.: Learning behavior in abstract memory schemes for dynamic optimization problems. Soft Comput. 13(12), 1163–1173 (2009)

[31] Rocha, M., Neves, J., Veloso, A.: Evolutionary algorithms for static and dynamic optimization of fed-batch fermentation processes. In: Ribeiro, B., et al. (eds.) Adaptive and Natural Computing Algorithms, pp. 288–291. Springer (2005)

[32] Singh, H.K., Isaacs, A., Nguyen, T.T., Ray, T., Yao, X.: Performance of infeasibility driven evolutionary algorithm (IDEA) on constrained dynamic single objective optimization problems. In: Proc. 2009 IEEE Congr. Evol. Comput., pp. 3127–3134 (2009)

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

[34] Thompson, J.M., Dowsland, K.A.: A robust simulated annealing based examination timetabling system. Comput. Oper. Res. 25(7-8), 637–648 (1998)

[35] Wang, Y., Wineberg, M.: Estimation of evolvability genetic algorithm and dynamic environments. Genetic Programming and Evolvable Machines 7(4), 355–382 (2006)

[36] Williams, K.P.: Simple genetic algorithm (SGA) source code (in C), `http://www.kenwilliams.org.uk/code/ga2.c` (accessed December 2008)

[37] Yang, S.: Genetic algorithms with memory- and elitism-based immigrants in dynamic environments. Evol. Comput. 16(3), 385–416 (2008)

[38] Yang, S., Yao, X.: Experimental study on population-based incremental learning algorithms for dynamic optimization problems. Soft Comput. 9(11), 815–834 (2005)

[39] Zhang, Z., Liao, M., Wang, L.: Multi-objective immune genetic algorithm solving dynamic single-objective multimodal constrained optimization. In: Proc. 8th Int. Conf. Natural Comput., pp. 864–868 (2012)