Chapter 11
Artificial Immune System for Solving Dynamic Constrained Optimization Problems

Victoria S. Aragón, Susana C. Esquivel, and Carlos A. Coello

Abstract. In this chapter, we analyze the behavior of an adaptive immune system when solving dynamic constrained optimization problems (DCOPs). Our proposed approach is called Dynamic Constrained T-Cell (DCTC) and it is an adaptation of an existing algorithm, which was originally designed to solve static constrained problems. Here, this approach is extended to deal with problems which change over time and whose solutions are subject to constraints. Our proposed DCTC is validated with eleven dynamic constrained problems which involve the following scenarios: dynamic objective function with static constraints, static objective function with dynamic constraints, and dynamic objective function with dynamic constraints. The performance of the proposed approach is compared with respect to that of another algorithm that was originally designed to solve static constrained problems (*SMES*) and which is adapted here to solve DCOPs. Besides, the performance of our proposed DCTC is compared with respect to those of two approaches which have been used to solve dynamic constrained optimization problems (RIGA and dRepairRIGA). Some statistical analysis is performed in order to get some insights into the effect that the dynamic features of the problems have on the behavior of the proposed algorithm.

Victoria S. Aragón · Susana C. Esquivel
Laboratorio de Investigación y Desarrollo en Inteligencia Computacional (LIDIC),
Universidad Nacional de San Luis - Ejército de los Andes 950 (5700)
San Luis, Argentina
e-mail: vsaragon@unsl.edu.ar

Carlos A. Coello
CINVESTAV-IPN (Evolutionary Computation Group) - Computer Science Department,
Av. IPN No. 2508, Col. San Pedro Zacatenco,
México D.F. 07300, México
e-mail: ccoello@cs.cinvestav.mx

E. Alba et al. (Eds.): Metaheuristics for Dynamic Optimization, SCI 433, pp. 225–263.
springerlink.com

11.1 Introduction

Three are the main scenarios that we could have in dynamic constrained optimization problems (DCOPs): 1) we could have a dynamic objective function and static constraints (if the objective function changes over time, this could affect the location of the optimum, which could move, for example, from a disconnected feasible region to another one), 2) we could have a static objective function and dynamic constraints (in this case, new local optima (or even a new global optimum) could appear as the infeasible region changes), and 3) we could have a dynamic objective function and dynamic constraints (this is perhaps the most difficult case, since there are changes in both the location of the optimum and the infeasible region).

Regardless of the scenario that we consider, DCOPs are, clearly, very difficult problems [17]. When the objective function is moving, it is necessary to have good mechanisms to track it. When the dynamic components of the problem are given by the constraints, then the changes could vary the shape, ratio (with respect to the entire search space), and/or structure of the feasible region [17].

DCOPs are not simply an academic challenge, since there are several real-world problems with such features. For example: the cargo movement problem in metropolitan areas adjacent to marine ports. In particular, truck scheduling and route planning, where ISO (International Standards Organization) containers need to be transferred between marine terminals, intermodal facilities, and end customers. In such an application, the objective is to reduce empty miles, and to improve customer service. A dynamic component is given in this case for incorporating the information of new customers after the set of routes has been determined [10]. Another example is the assembly of a schedule for transport ships, where the ships transport liquified natural gas from different ports around the world to one destination port. In this case, after finding a valid schedule, a recalculation may be needed because some ships got delayed (for example, due to a storm or some mechanical damage) [14]. Another example are the hydro-thermal power generation systems, in which both the hydroelectric and the thermal generating units are utilized to meet the total power demand. The optimum power scheduling problem involves the allocation of power to all concerned units, but the total fuel cost of thermal generation and emission properties has to be minimized, while satisfying all the constraints imposed by the hydraulic and power system networks. The problem is dynamic due to the changing nature of power demand over time. Thus, ideally, the optimal power scheduling problem should be considered as an online dynamic optimization problem in which solutions must be found when there is a change in the power demand [9]. All of these problems have a great industrial impact, and their efficient solution can, therefore, be very profitable. Surprisingly enough, however, the literature on the solution of DCOPs is relatively scarce. In this chapter, we propose the use of an algorithm based on an adaptive immune system model for solving DCOPs. The proposed approach has been inspired by the immune responses mediated by the T cells, and constitutes an extension of an algorithm (developed by the authors of this chapter) that was originally designed for solving static constrained optimization problems.

The remainder of this chapter is organized as follows. Section 11.2 formally defines the problem of our interest. In Section 11.3, we provide a review of the previous related work. In Section 11.4, we describe our proposed approach. Section 11.5 presents the test problems and performance measures adopted to evaluate our proposed approach. This section also describes the algorithm that we used to compare our results and the experimental design that we adopted. Finally, Sections 11.6 and 11.7 present the results that we obtained from the experiments performed and our conclusions, respectively.

11.2 Problem Statement

We are interested in solving problems of the form[1]:

minimize

$$f(X;t) \tag{11.1}$$

subject to:

$$g_j(X;t) \leq 0 \qquad j = 1,\ldots,m \tag{11.2}$$

$$h_k(X;t) = 0 \qquad k = 1,\ldots,p \tag{11.3}$$

$$x_i^l \leq x_i \leq x_i^u \qquad i = 1,\ldots,n. \tag{11.4}$$

In Equation (11.1), f designates the objective function, $X = (x_1, x_2, \ldots, x_n)^T$ is a vector containing the design variables, and t is a positive integer which denotes time. The remaining functions correspond to inequality constraints g (Equation (11.2)), equality constraints h (Equation (11.3)), and side constraints with lower and upper limits indicated by the superscripts l and u (Equation (11.4)), respectively. Both the objective function and the constraints could be linear or nonlinear.

When an inequality constraint takes a zero value at the optimum, we say that it is **active**. By definition, all equality constraints are active at all points of the search space.

11.3 Previous Related Work

The literature on the solution of DCOPs using artificial immune systems is very scarce and is briefly reviewed next. Schiex et al. [29] defined the maintenance solution problem in dynamic constraint satisfaction problems (CSPs) and underlined that the iterative application of commonly used constraint satisfaction algorithms would result in redundant search and inefficiency. They also indicated that a complete description of the space explored justified in terms of the set of constraints may

[1] Without loss of generality, we will assume only minimization problems.

grow exponentially in space. Thus, they proposed a class of nogood recording algorithms to solve the satisfaction problem and simultaneously offered a polynomially bounded space compromise between both of these approaches. They outperformed all algorithms considered for the solution maintenance problem in dynamic CSPs, and also provided very good results for static CSPs.

Modi et al. [16] proposed a formalization of distributed resource allocation that, its authors claim to be expressive enough to represent both dynamic and distributed aspects of the problem. They defined different categories of difficulties of the problem and presented complexity results for them. Also, they defined the notion of Dynamic Distributed Constraint Satisfaction Problem (DyDCSP) and presented a generalized mapping from distributed resource allocation to DyDCSP. Through both theoretical analysis and experimental verifications, they showed that this approach to dynamic and distributed resource allocation is powerful and can be applied to real-world problems such as the Distributed Sensor Network Problem.

Mailler [11] presented two protocols for solving dynamic distributed constraint satisfaction problems which are based on the classical Distributed Breakout Algorithm (DBA) and the Asynchronous Partial Overlay (APO) algorithm. These two new algorithms are compared on a broad class of problems varying their overall difficulty as well as the rate at which they change over time. The results obtained by the author indicate that neither of the algorithms completely dominates the other on all problem classes, but that depending on environmental conditions and the needs of the user, one method may be preferable over the other.

Richter and his collaborators have investigated the use of evolutionary algorithms in dynamic environments in a number of papers [19–28]. Some of these works will be briefly reviewed next.

In [19], Richter studied the behavior of an evolutionary algorithm in dynamic environments that change chaotically. Additionally, he analyzed the concept of dynamic severity when applied to chaotic changes, as well as the relationship between severity, change frequency, and predictability of the changes. Richter et al. [20] proposed a memory scheme based on abstraction for evolutionary algorithms with the aim of solving dynamic optimization problems. In this scheme, the memory does not store good solutions as themselves but as their abstraction, i.e., their approximate location in the search space. Thus, when the environment changes, the stored abstraction information is extracted to generate new individuals into the population. The authors argued that their results show the efficiency of their proposed approach. In a further paper, Richter [25] proposed a memory design for solving constrained dynamic optimization problems using an evolutionary algorithm. Based on ideas from abstract memory, Richter introduced and tested two schemes: blending and censoring. Through some experiments he showed that such a memory can be used to solve certain types of constrained dynamic problems. Richter's work also involves a study of the automatic detection of changes through population-based and sensor-based schemes [22]. In another paper, Richter [24] employed a negative selection algorithm to detect changes in order to solve dynamic optimization problems.

His numerical experiments showed that the use of an immunological approach can be successfully used to solve the change detection problem for dynamic fitness landscapes. In Richter et al. [26], the authors proposed a method for generating variable-sized detectors in the framework of negative selection based artificial immune systems. The method is inspired by the idea of interpreting the feature space as a potential field. The authors used a divide-and-conquer algorithm in order to accelerate the generation of detectors. They also generalized the idea of overlapping detectors by introducing multiple detector layers compromising different geometric structures for the used detectors. In [27], Richter et al. proposed a method for solving the change detection problem for constrained dynamic optimization using evolutionary algorithms. The basis for this detection is the use of both fitness values and corrected fitness values of the individuals, without requiring any additional data from the fitness landscape. The fitness distribution of the individuals of one generation is analyzed using simple statistical measures and by hypothesis test based classifiers. The authors argued that good results could be found for the constraint change and the landscape change detection, for classifiers based on the Kolmogorov-Smirnov test whereas simple statistical methods failed to detect changes with high robustness. Richter et al. [28] considered optimization problems with a dynamic fitness landscape and dynamic constraints that may change in an independent manner. The authors argued that this situation can lead to asynchronous change patterns with the possibility of occasional synchronization points. So, they presented a framework for describing such a dynamical setting and for performing numerical experiments on the algorithm's behavior. The DynCOAA algorithm was proposed by Mertens et al. [14]. It is based on the Ant Colony Optimization (ACO) metaheuristic and was designed for solving DCOPs. Its authors compared this approach with respect to two other algorithms in two different types of problems: artificial graph coloring problems and real-world ship scheduling problems. The main conclusions from the authors were the following: DynCOAA is well suited for solving DCOPs, as it beats DynAWC [16, 29] in the two types of problems that they studied and DynDBA [11] in one of them. Their second conclusion was that choosing the right algorithm for a specific problem is very important because the performance of the algorithms greatly depends on the type of problem being solved. The main difference between DynCOAA and the approach proposed by us in this paper resides on the biological metaphors they are based on: DynCOAA is based on the behavior of ants when foraging for food and ours is based on the behavior of the immune system when receiving an external attack. There is also another difference worth emphasizing. DynCOAA builds a solution and then it takes the best solution as a guideline when a change occurs. In contrast, our approach does not follow the direction of the best solution found so far but, instead, the activities of each cell are influenced by another random cell from the same population to which the cell belongs.

Deb et al. [9] modified the NSGA-II so that it could track down a new Pareto optimal front, as soon as there was a change in the problem. The authors investigated the introduction of a few randomly generated solutions or a few mutated solutions. The proposed approaches were tested and compared on a benchmark problem and

on the real-world optimization of a hydro-thermal power scheduling problem. This systematic study was able to find a minimum frequency of change allowed in the problem for two dynamic EMO procedures to adequately track down the Pareto optimal frontiers on-line. Based on their results, the authors suggested an automatic decision-making procedure for arriving at a single optimal solution on-line.

Nguyen et al. [17] studied the characteristics that might make dynamic constrained problems difficult to solve by some of the existing dynamic optimization and constraint-handling algorithms. They also introduced a set of (numerical) dynamic benchmark problems with the features analyzed in the paper. They tested several dynamic and constrained optimization strategies on the proposed benchmark problems, including the use of two canonical algorithms, a triggered hyper-mutation Genetic Algorithm (GA), a random-immigrant GA named RIGA (*RIGA-elit* is an algorithm derivated from the GA (Genetic Algorithm), but after applying the mutation operator on each generation, a fraction of the population is replaced by randomly generated individuals, in order to maintain diversity.), and a GA plus repair, named *dRepairGA*. The authors stated that their results confirm that dynamic constrained problems have special characteristics that might significatively affect the performance of the algorithms traditionally used for static problems. At the end of the paper, they proposed a list of possible requirements that an algorithm should meet to solve dynamic constrained problems effectively. Our algorithm differs from the ones used by Nguyen et al. [17] in the following aspects: in our proposed DCTC, the mutation probability is not increased when a change occurs, but instead, it remains fixed during all the search process. Additionally, any of the randomly generated cells is inserted into any population, except when virgin cells are initializated. Finally, our proposed DCTC works over feasible as well as over infeasible solutions and, therefore, it does not require any repair algorithm. Nguyen et al. [19] proposed a new approach to solving DCOPs, combining existing dynamic optimization techniques with constraint-handling techniques in order to handle objective-function changes and constraint-function changes separately and differently. They modified an existing repair method to track down the moving constraints and combined it with existing random-immigrant and hyper-mutation operators, in order to handle objective-function changes. They also used different techniques to detect objective function changes and constraint-function changes separately and differently. This proposed approach was used to derive two new algorithms. The first of them was called *dRepairGA* and is based on *dRepairRIGA*, an algorithm which integrates the characteristics of a GA, a repair method (to transform infeasible solutions into feasible ones, if possible) and the dynamic optimization strategy *RIGA-elit*). The second approach was called *dGenocop* and is based on Genocop III. They also proposed variants of these two algorithms. The authors validated their algorithms using 18 test problems and argued that their proposed algorithms were able to significantly outperform GA/RIGA/HyperM and GA+Repair/Genocop III in solving DCOPs while still maintaining equal or better overall performance in solving other groups of problems except for the static cases.

11.4 Our Proposed Approach

Our proposed approach consists of an adaptive immune system model based on the immune responses mediated by the T cells. Originally, this approach was used to solve static optimization problems (see [3]). Then, it was extended to solve dynamic (unconstrained) problems and, later on, to solve (static) constrained problems (see [1, 2, 4, 5]).

The model that we developed is called TCELL, and it considers many of the processes that T cells suffer from their origin in the hematopoietic stem cells in the bone marrow until they become memory cells. T cells belong to a group of white blood cells known as lymphocytes. They play a central role in cell-mediated immunity. They present special receptors on their cell surface called T cell receptors (TCR[2]). All T cells originate from hematopoietic stem cells in the bone marrow. Hematopoietic progenitor derived from hematopoietic stem cells populate the thymus and expand by cell division to generate a large population of immature thymocytes [30].

Several subsets of the T cells have been discovered, each with a distinct function. Thus, they can be classified in different populations according to the antigen receptor they express. These antigens receptors could be TCR-1 or TCR-2. Additionally, TCR-2 cells express CD4 or CD8.[3]

Also, T cells can be divided into three groups according to their maturation or development level (phylogenies of the T cells [8]): virgin, effector, and memory cells. Virgin cells are those which have never been activated (i.e., they have not suffered proliferation or differentiation). At the beginning, these cells do not express CD4, nor CD8. However, later on, they develop and express both marks, CD4 and CD8. Finally, virgin cells mature and express only one mark, either CD4 or CD8. Before these cells release the thymus, they are subject to both positive selection [12] and negative selection [12]. Positive selection guarantees that the only survivors are the cells with TCRs that present a moderate affinity with respect to the self MHC. Negative selection eliminates the cells with TCRs that recognize self components unrelated to the MHC.

Effector cells are a type of cells that express only one mark, CD4 or CD8. They can be activated by co-stimulating signals plus their ability to recognize an antigen [7, 13]. The immune cells interact through the secretion of cytokines.[4] Cytokines allow cellular communication. Thus, an immune cell c_i influences the activities (proliferation and differentiation) of another cell c_j through the secretion of cytokines, modulating the production and secretion of cytokines by c_j [8]. In order to activate an effector cell, a co-stimulated signal is necessary. Such signal corresponds to the cytokines secreted from another effector cell. The activation of an effector cell im-

[2] TCRs are responsible for recognizing antigens bound to major histocompatibility complex (MHC) molecules.

[3] Lymphocytes express a large number of surface molecules that can be used to mark different cellular populations. CD means *Cluster Denomination* and indicates the group to which lymphocytes belong.

[4] Proteins act as signal transmitters between cells, and also induce growth, differentiation, activation, etc.

plies that it will be replicated and differentiated. Thus, the proliferation process has as its goal to replicate the cells and the differentiation process changes the clones so that they acquire specialized functional properties.

Finally, the memory cells are those that remain in the host even when the infection or danger has been overtaken, so that in the future, they are able to get stimulated by the same or by a similar antigen. Usually, they respond (through proliferation and differentiation) faster with a low dosage of antigens than the B memory cells. It is worth noting that, although the effector and memory cells are replicated, they are not subject to somatic hypermutation. For the effector cells, the differentiation process is subject to the cytokines released by another effector cell. In our model, the differentiation process of the memory cells relies on their own cytokines. The immune response consists of two phases: the first (called *recognizing phase*) involves the processes that suffer only the virgin cells and the second (called *effector phase*) is related to the processes that suffer the effector and memory cells. The *recognizing phase* has to provide some diversity so that the next phase can produce a cell to eliminate the antigen. Meanwhile, the *effector phase* is in charge of doing this job.

Summarizing the features of the natural immune system that inspired our model, we can highlight that TCELL considers that T cells react when the system is invaded by an external pathogen as well as the presence of co-stimulating signals, sent by the own T cells, according to the Danger Theory. Additionally, TCELL uses the Self-Non-self concept (in the *recognizing phase*) in order to remove undesirable cells which can be considered dangerous to the host. Finally, TCELL also considers the interaction among the T cells through the secretion of cytokines, as a communication mechanism.

11.4.1 Proposed Algorithm Based on T-CELL

DCTC (Dynamic Constrained T-Cell) is an algorithm inspired on the TCELL model [4], which we propose here to solve dynamic constrained optimization problems. DCTC operates on four populations, corresponding to the groups in which the T-cells are divided: (1) Virgin Cells (VC), (2) Effector Cells with cluster denomination CD4 (CD4), (3) Effector Cells with cluster denomination CD8 (CD8), and (4) Memory Cells (MC). Each population is composed by a set of T cells whose characteristics are subject to the population to which they belong.

Virgin Cells (VC) do not suffer the activation process. They have to provide diversity. This is reached through the random acquisition of TCR receptors. Virgin cells are represented by: 1) a *TCR* represented by a bitstring using Gray coding (called TCR_b) and 2) a *TCR* represented by a vector of real numbers (called TCR_r).

Into the natural immune system, the positive and negative selections have to remove the potentially harmful cells. Thus, in our proposed algorithm, positive selection is in charge of eliminating the cells that recognize the antigen with a low matching. On the other hand, negative selection has to eliminate the cells that have a similar TCR, according to a Hamming or a Euclidean distance, depending on whether the TCR is represented by a TCR_b or by a TCR_r.

Effector Cells are composed by: 1) a TCR_b or TCR_r, if they belong to CD4 or CD8, respectively, 2) a proliferation level, and 3) a differentiation level. The goal of this type of cell is to explore in a global way the search space. Thus, CD4 explores the search space, taking advantage of the Gray coding properties (there is only one bit of difference between two consecutive numbers), while CD8 uses real numbers representation (big or small jumps).

The goal of the memory cells is to explore the neighborhood of the best found solutions. These cells are represented by the same components as CD8.

In our proposal, the TCR identifies the decision variables of the problem, independently of the TCR representation. The proliferation level indicates the number of clones that will be assigned to a cell and the differentiation level indicates the number of bits or decision variables (depending on the TCR representation adopted) that will be changed, when the differentiation process is applied.

The activation of an effector cell, called ce_i, implies the random selection of a set of potential activator (or stimulating) cells. The closest cell to ce_i (using Hamming or Euclidean distance), according to the TCR in the set, is chosen to become the stimulating cell, say ce_j. Then, ce_i proliferates and differentiates.

At the beginning, the proliferation level of each stimulated cell, ce_i, is given by a random value within [1, 3],[5] but then, it is determined taking into account the proliferation level of its stimulating cell (ce_j). If the ce_i is better than ce_j, then ce_i keeps its own proliferation level; otherwise, ce_i receives a level which is 10% lower than the level of ce_j.

Memory cells proliferate and differentiate according to their proliferation level (randomly between 1 and the size of MC[6]) and differentiation level (number of decision variables,[7]) respectively. Both levels are independent from the other memory cells.

In our proposed DCTC algorithm, the constraint-handling method needs to calculate, for each cell (solution), regardless of the population to which it belongs, the following: 1) the sum of constraint violations (sum_res)[8] and 2) the value of the objective function (only if the cell is feasible).

We consider that a ce_i cell is better than a ce_j cell if: 1) TCR's ce_i is feasible and TCR's ce_j is infeasible, 2) both cells have feasible TCRs but the objective function value of ce_i is lower than the objective function value of ce_j, and 3) both cells have infeasible TCRs but sum_res' of ce_i is lower than sum_res' of ce_j. This criterion is used to sort the population. Each type of cell has its own differentiation process, which is blind to their representation and population:

Differentiation for CD4: the differentiation level of ce_i is determined by the Hamming distance between the stimulated (ce_i) and stimulating (ce_j) cells. It indicates the number of bits to be changed. Each decision variable and the bit to

[5] This value was derived after numerous experiments.

[6] This is an arbitrary value in order to avoid overloading the number of required parameters.

[7] This value was set thinking on performing an intensive local search.

[8] This is a positive value determined by $g_i(x)^+$ for $i = 1, \ldots, m$ and $|h_k(x)|$ for $k = 1, \ldots, p$.

be inverted are chosen in a random way. The bits change according to a probability $\text{prob}_{diff-CD4}$. The pseudo-code for the proliferation and differentiation of cell ce_i is shown in Algorithm 11.1.

Algorithm 11.1. Differentiation CD4

```
for np = 1 to Proliferation Level of ce_i do
    clone^np ← ce_i ;
end
for nd = 1 to Differentiation Level of ce_i do
    if prob_diff−CD4 then
        k ← U(1, | vd |);
        l ← U(1, | bits_k |);
        Invert the l^th-bit of vd_k of the clone^np;
    end
end
```

where $U(1,w)$ refers to a random number with a uniform distribution in the range (1,w), $\mid vd \mid$ is the number of decision variables of the problem, $\mid bits_k \mid$ is the number of bits to represent the k^{th} decision variable, and vd_k indicates the k^{th} decision variable.

Differentiation for CD8: the differentiation level for cell ce_i is related to its stimulating cell (ce_j). If the TCR_r of the ce_j is better than the TCR_r of the stimulated cell ce_i (according to the objective function value), then the level (for ce_i) is a random number within [1, $\mid dv \mid$[9]; otherwise, it is a random value within [1, $\mid dv \mid /2$], where $\mid dv \mid$ is the number of decision variables of the problem. Each variable to be changed is chosen in a random way and it is modified according to Equation (11.5):

$$x' = x \pm \frac{U(0, lu - ll)^{U(0,1)}}{10^7 iter}, \tag{11.5}$$

where x and x' are the original and the mutated decision variables, respectively. lu and ll are the upper and lower bounds of x, respectively. $iter$ indicates the number of iterations until reaching the maximum number of evaluations for a change. At the moment of the differentiation of a cell (ce_i), the value of the objective function of its stimulating cell (ce_j) is taken into account. In order to determine if $r = \frac{U(0,lu-ll)^{U(0,1)}}{10^7 iter}$, will be added or subtracted to x, the following criteria are considered: if ce_j is better than ce_i and the decision variable value of ce_j is less than the value of ce_i, or if ce_i is better than ce_j and the decision variable value of ce_i is less than the value of ce_j, then r is subtracted from x; otherwise, r is added to x. Both criteria aim to guide the search towards the best solutions found so far. The pseudo-code for the proliferation and differentiation of the cell ce_i with the stimulating cell ce_j is shown in Algorithm refCD8:

[9] If the stimulating cell is better, then ce_i should change more decision variables

Algorithm 11.2. Differentiation CD8

```
for np = 1 to Proliferation Level of ce_i do
    Clone^np ← ce_i ;
    for nd = 1 to Differentiation Level of ce_i do
        k ← U(1, | vd |);
        r ← (U(0, lu − ll) / 10^7 iter)^U(0,1) ;
        if f(ce_jTCR_r) is better than f(ce_iTCR_r) and ce_jTCR_rk < ce_iTCR_rk o f(ce_iTCR_r) is
        better than f(ce_jTCR_r) and ce_jTCR_rk > ce_iTCR_rk then
            Clone^np TCR_rk ← ce_iTCR_rk − r
        else
            else if f(ce_jTCR_r) is better than f(ce_iTCR_r) and ce_jTCR_rk > ce_iTCR_rk o
            f(ce_iTCR_r) is better than f(ce_jTCR_r) and ce_jTCR_rk < ce_iTCR_rk then
                Clone^np TCR_rk ← ce_iTCR_rk + r;
            end
            Add or subtract r with probability 50%;
        end
    end
end
```

where $U(w_1, w_2)$ refers to a random number with a uniform distribution in the range (w_1,w_2), $\mid vd \mid$ is the number of decision variables of the problem, lu_x and ll_x are the upper and lower bounds of x, respectively. $iter$ indicates the number of iterations until reaching the maximum number of evaluations for a change. $f(ce_{hTCR_r})$ is the objective function value for the TCR$_r$ of the cell ce_h, and $ce_{hTCR_{r_k}}$ indicates the k^{th} decision variable of the cell h. If after ten trials, the procedure cannot find an x' in the allowable range, a random number with a uniform distribution is assigned to it.

Differentiation for MC: the number of decision variables to be changed is determined by the differentiation level of the cell to be differentiated. Each variable to be changed is chosen in a random way and it is modified according to Equation (11.6):

$$x' = x \pm \left(\frac{U(0, lu_x - ll_x)}{10^7 iter} \right)^{U(0,1)}, \tag{11.6}$$

where x and x' are the original and the mutated decision variables, respectively. $U(0, w)$ refers to a random number with a uniform distribution in the range (0,w). lu_x and ll_x are the upper and lower bounds of x, respectively. $iter$ indicates the number of iterations until reaching the maximum number of evaluations for a change. In a random way, we decide if $r = \left(\frac{U(0, lu_x - ll_x)}{10^7 iter} \right)^{U(0,1)}$ will be added or subtracted to x. If after ten trials, the procedure cannot find an x' in the allowable range, then a random number with a uniform distribution is assigned to it.

The general structure of our proposed algorithm for dynamic constrained optimization problems is given in Algorithm 11.3.

Algorithm 11.3. DCTC Algorithm

```
1: Initialize_VC();
2: Evaluate_VC();
3: Assign_Proliferation();
4: Divide_CDs();// take into account feasibility of the solutions
5: Positive_Selection_CD4();// eliminate the worst cells in CD4
6: Positive_Selection_CD8();// eliminate the worst cells in CD8
7: Negative_Selection_CD4();// eliminate the most similar cells in CD4
8: Negative_Selection_CD8();// eliminate the most similar cells in CD8
9: while A predetermined number of changes has not been reached do
10:    while A predetermined number of evaluations has not been performed do
11:       Proliferate_CD4();
12:       Differentiate_CD4();
13:       Sort_CD4();
14:       Proliferate_CD8();
15:       Differentiate_CD8();
16:       Sort_CD8();
17:       Insert_CDs_en_MC();
18:       for i = 1 to rep_MC do
19:          Proliferate_MC();
20:          Differentiate_MC();
21:       end for
22:       Sort_CM();
23:    end while
24:    Statistics();
25:    Change_Function();
26:    Re-evaluate_Populations();
27: end while
```

The algorithm works in the following way. At the beginning, the TCR_b and TCR_r from the virgin cells are initialized in a random way, according to the TCR's encoding (step 1). Then, each TCR of a virgin cell is evaluated (step 2). In step 3, the proliferation levels are assigned. Then, in step 4, the virgin cells are divided taking into account their feasibility. Next, the feasible cells, TCR_b and TCR_r, from VC are selected to form CD4 and CD8, respectively. If it is not possible to complete the required number of cells, then the infeasible TCRs needed to reach such a value are selected. Each effector cell will inherit the proliferation level of the virgin cell which received the TCR.

The negative and positive selections are applied to each effector population (CD4 and CD8). The first selection eliminates 10% of the worst cells and the second

selection eliminates cells that are similar among them (keeping the best from them). This mechanism works in the following way: for each effector cell, we search inside its population the closest cell (using Hamming or Euclidean distance according to the TCR's cell) and the worst from them is eliminated. This process reduces the effector's population sizes.

The first iteration (line 9) is controlled by the number of changes of the objective function.

Furthermore, for each change, a maximum number of objective function evaluations is allowed (line 10)[10]. The steps inside the last iteration are: first, to activate the CD4 population; in other words, to perform proliferation and differentiation of all the cells from CD4 (lines 11 and 12). Then, these cells are sorted (line 13). Next, the CD8 population is activated. This means that we perform proliferation and differentiation of all the cells from CD8 (lines 14 and 15), which are sorted in (line 16).

The best solutions from CD4 and CD8 are inserted or are used to replace the 50% of the worst solutions in MC (depending on whether or not, MC is empty) (lines 17). Since the representation schemes of the TCR, for CD4 and MC, are different, before the insertion of the best cell from CD4 (with TCR_b) into MC, the receptor has to be converted into a real-values vector (TCR_r). For this process, we use Equation (11.7), which takes as input a bitstring generated with Gray coding and returns a real number (this process is applied as many times as decision variables has the problem):

$$dv_j = l_{lj} + \frac{\sum_{i=0}^{L_j} 2^{L_j - i} dv'_{ij}(lu_j - ll_j)}{2_j^L - 1}, \tag{11.7}$$

where dv_j is the j^{th} decision variable with $j = 1, \ldots,$ number of decision variables of the problem, L_j is the number of bits for the j^{th} decision variable, lu_j and ll_j are the upper and lower limits for the decision variable dv_j, respectively. And dv'_{ij} is the i^{th} bit of the bitstring that represents dv_j. Also, Equation (11.7) is used when a cell from CD4 has to be decoded in order to be evaluated. Next, the cells from MC are activated a certain (predefined) number of times, rep_{MC} (lines 19 and 20).

The algorithm is notified about the existence of a change in the environment (line 25), since that information is required in order to re-evaluate the populations (step 26). Even when some literature about change detection exists (see for example, [22, 24, 26, 27]), dealing with this (rather difficult) topic is beyond the scope of this chapter. Here, we only focus on the mechanisms to react to any incoming changes. In fact, when using metaheuristics, it is normally assumed that the search engine will be informed whenever a change has occurred.

[10] Since it is not known *a priori* how many clones will be assigned to each cell, it is possible to exceed the maximum number of evaluations per change in $3 \mid feasible(CD4) \mid + \mid feasible(CD8) \mid + rep_{MC} \mid feasible(MC) \mid^2$, where $feasible(x)$ indicates the number of feasible solutions in population x.

11.5 Experiments

In this section, we describe our experimental setup. This includes a description of the set of test problems used to validate our proposed DCTC, the performance measures used to evaluate it, the corresponding parameters settings, and the description of the algorithm chosen to compare our results. Some statistical analysis is performed in order to determine the effect of the dynamic features of the problems on the behavior of the proposed algorithm.

11.5.1 Dynamic Constrained Benchmark

In order to validate our proposed approach, we adopted eleven dynamic constrained optimization problems from a set that was originally proposed by Nguyen et al. [17]. The subset of problems chosen presents some kind of dynamism, either in the objective function, in the constraint or both. Table 11.1 summarizes the main features of the test problems adopted.

11.5.2 Performance Measures

Here, we describe the performance measures adopted for our experimental study. One of them is relatively popular in the literature [6]: the **offline error** (*oe*), which represents the average of the best error at each iteration. This measure is calculated here, only for feasible solutions. If an infeasible solution is found then nothing is added, as defined by Equation (11.8).

$$oe = \frac{1}{N_c} \sum_{j=1}^{N_c} \sum_{i=1}^{iter} (f_j^* - f_{ji}^*)), \tag{11.8}$$

where N_c is the total number of changes within an experiment, $iter$ is the current iteration number, f_j^* is the value of the optimum solution for the jth state[11] and f_{ji}^* is the current best fitness value found for the j^{th} state.

The ideal value for *oe* is zero, which would mean that the optimum was found at the very beginning of each state.

As *oe* is calculated only for feasible solutions, it is possible that this value becomes zero or a value close to it, but without finding any feasible solution. For this reason, we use this measure along with the measure rf, which calculates the percentage of runs in which at least one feasible solution was found for all the changes that took place. Thus, the ideal value of rf is 100%, which would mean that, in all runs, feasible solutions were found, for all changes.

[11] We call *state* to the time period where objective function and constraints remain fixed.

Table 11.1 Main features of the test problems adopted

Problem	ObjFunc	Constr.	DFR	Parameters Setting
G24_1	f^1 Dynamic	g^1 g^2 Fixed	2	$p_2(t) = r_i(t) = 1; q_i(t) = s_i(t) = 0; p_1(t) = sin(k\pi t + \frac{\pi}{2})$
G24_2	f^1 Dynamic	g^1 g^2 Fixed	2	$if(t\,mod\,2 = 0) = \begin{cases} p_1(t) = sin(\frac{k\pi t}{2} + \frac{\pi}{2}) \\ p_2(t) = \begin{cases} p_2(t-1) & if\ t > 0 \\ p_2(0) = 0 & if\ t = 0 \end{cases} \end{cases}$ $if(t\,mod\,2 \neq 0) = \begin{cases} p_1(t) = sin(\frac{k\pi t}{2} + \frac{\pi}{2}) \\ p_2(t) = sin(\frac{k\pi(t-1)}{2} + \frac{\pi}{2}) \end{cases}$ $r_i(t) = 1; q_i(t) = s_i(t) = 0;$
G24_3	f^1 Fixed	g^1 g^2 Dynamic	2-3	$p_i(t) = r_i(t) = 1; q_i(t) = s_1(t) = 0; s_2(t) = 2 + t\frac{x_2\max - x_2\min}{S}$
G24_3b	f^1 Dynamic	g^1 g^2 Dynamic	2-3	$p_1(t) = sin(k\pi t + \frac{\pi}{2}); p_2(t) = r_i(t) = 1; q_i(t) = s_1(t) = 0; s_2(t) = 2 + t\frac{x_2\max - x_2\min}{S}$
G24_4	f^1 Dynamic	g^1 g^2 Dynamic	2-3	$p_1(t) = sin(k\pi t + \frac{\pi}{2}); p_2(t) = r_i(t) = 1; q_i(t) = s_1(t) = 0; s_2(t) = t\frac{x_2\max - x_2\min}{S}$
G24_5	f^1 Dynamic	g^1 g^2 Dynamic	2-3	$if(t\,mod\,2 = 0) = \begin{cases} p_1(t) = sin(\frac{k\pi t}{2} + \frac{\pi}{2}) \\ p_2(t) = \begin{cases} p_2(t-1) & if\ t > 0 \\ p_2(0) = 0 & if\ t = 0 \end{cases} \end{cases}$ $if(t\,mod\,2 \neq 0) = \begin{cases} p_1(t) = sin(\frac{k\pi t}{2} + \frac{\pi}{2}) \\ p_2(t) = sin(\frac{k\pi(t-1)}{2} + \frac{\pi}{2}) \end{cases}$ $r_i(t) = 1; q_i(t) = s_1(t) = 0; s_2(t) = t\frac{x_2\max - x_2\min}{S}$
G24_6a	f^1 Dynamic	g^3 g^6 Fixed	2	$p_1(t) = sin(\pi t + \frac{\pi}{2}); p_2(t) = r_i(t) = 1; q_i(t) = s_i(t) = 0;$
G24_6c	f^1 Dynamic	g^3 g^4 Fixed	2	$p_1(t) = sin(\pi t + \frac{\pi}{2}); p_2(t) = r_i(t) = 1; q_i(t) = s_i(t) = 0;$
G24_6d	f^1 Dynamic	g^5 g^6 Fixed	2	$p_1(t) = sin(\pi t + \frac{\pi}{2}); p_2(t) = r_i(t) = 1; q_i(t) = s_i(t) = 0;$
G24_7	f^1 Fixed	g^1 g^2 Dynamic	2	$p_i(t) = r_i(t) = 1; q_i(t) = s_1(t) = 0; s_2(t) = t\frac{x_2\max - x_2\min}{S}$
G24_8b	f^2 Dynamic	g^1 g^2 Fixed	2	$p_i(t) = -1; q_i(t) = -(1.4706 + 0.859cos(k\pi t)); q_2(t) = -(3.442 + 0.859sin(k\pi t)); r_i(t) = 1; s_i(t) = 0$

Fixed - There is no change
Dynamic - The function is dynamic

$f^1 = -(X_1(x_1;t) + X_2(x_2;t))$

$f^2 = -3\exp\left(-\sqrt{\sqrt{(X_1(x_1;t))^2 + (X_2(x_2;t))^2}}\right)$

$g^1 = -2Y_1(x_1;t)^4 + 8Y_1(x_1;t)^3 - 8Y_1(x_1;t)^2 + Y_2(x_2;t) - 2$

$g^2 = -4Y_1(x_1;t)^4 + 32Y_1(x_1;t)^3 - 88Y_1(x_1;t)^2 + 96Y_1(x_1;t) + Y_2(x_2;t) - 36$

$g^3 = 2Y_1(x_1;t) + 3Y_2(x_2;t) - 9$

$g^4 = \begin{cases} -1 & if\ (0 \le Y_1(x_1;t) \le 1)\ or\ (2 \le Y_1(x_1;t) \le 3) \\ 1 & otherwise \end{cases}$

$g^5 = \begin{cases} -1 & if\ (0 \le Y_1(x_1;t) \le 0.5)\ or\ (2 \le Y_1(x_1;t) \le 2.5) \\ 1 & otherwise \end{cases}$

$g^6 = \begin{cases} -1 & if\ [(0 \le Y_1(x_1;t) \le 1)\ and\ (2 \le Y_2(x_2;t) \le 3)]\ or\ (2 \le Y_1(x_1;t) \le 3) \\ 1 & otherwise \end{cases}$

where $X_i(x;t) = p_i(t)(x + q_i(t))$, $Y_i(x;t) = r_i(t)(x + s_i(t))$, $0 \le x_1 \le 3$, $0 \le x_2 \le 4$, $p_i(t)$, $q_i(t)$, $r_i(t)$ and $s_i(t)$ are the dynamic parameters. The first two of them determine how the objective function changes over time and the rest determine how the constraint functions change
DFR - Number of Disconnected Feasible Regions
In all problems, except for G24_3 and G24_7, the global optimum switches between disconnected regions
Only in problem G24_3 a new optimum appears without changing the existing one.

From [17], we took the measures *ARR* and *RR*, which indicate how quickly does the algorithm converge to the global optimum before the next change occurs, and how quickly does the algorithm recover from an environmental change and starts converging to a new solution before a change occurs, respectively. *ARR* and *RR* are defined by equations (11.9) and (11.10):

$$ARR = \frac{1}{N_c} \sum_{i=1}^{N_c} \frac{\sum_{j=1}^{p(i)} [f_{ij}^* - f_{i1}^*]}{p(i)[f_i^* - f_{i1}^*]} \tag{11.9}$$

$$RR = \frac{1}{N_c} \sum_{i=1}^{N_c} \frac{\sum_{j=1}^{p(i)} [f_{ij}^* - f_{i1}^*)]}{p(i)[f_{ip(i)}^* - f_{i1}^*]} \tag{11.10}$$

where f_{ij}^* is the objective function value of the best feasible solution found since the last change until the jth iteration of the algorithm of the state i, N_c is the number of changes, $p(i)$ is the maximum number of iterations performed by the algorithm for the state i, and f_i^* is the optimum value for the state i. Both, *ARR* and *RR* have their ideal values in 1. Both *ARR* and *RR* would be 1 when the algorithm is able to recover and converge to a solution (the optimal solution for *ARR*) immediately after a change, and would be equal to zero in case the algorithm is completely unable to recover from the change.

Nguyen et al. [17] proposed how to analyze the convergence behavior/recovery speed of an algorithm through a plot of the *RR*/*ARR* scores. If a point is:

1. on the thick diagonal line, the algorithm has recovered and has converged to the optimum;
2. at the top right corner, the algorithm has recovered quickly and is having a good performance;
3. at the bottom right corner, it is likely that the algorithm has converged to a local optimum;
4. at the bottom left corner, the algorithm has recovered slowly and has not converged yet.

11.5.3 Parameters Settings

Since the literature on dynamic constrained optimization using artificial immune systems is scarce, in order to validate our proposed approach, we decided to adapt an algorithm that was originally proposed to solve (static) constrained optimization problems. This approach was proposed in [15], and it consists of a simple multimembered evolution strategy, called *SMES*. This approach does not require the use of penalty factors (or a penalty function at all). Instead, it uses a diversity mechanism based on allowing infeasible solutions to remain in the population. It also uses a comparison mechanism based on feasibility to guide the process towards the feasible region of the search space. Also, the initial step size of the evolution strategy is reduced in order to perform a finer search and a combined (discrete/intermediate)

panmictic recombination technique improves its exploitation capabilities. The approach was tested with a well-known benchmark, obtaining very competitive results. Its source code was taken from `http://www.cs.cinvestav.mx/~EVOCINV/SES/principal.html`. We modified this code by adding a mechanism for re-evaluating populations after a change occurs. We called this new version *SMESD*. Table 11.2 highlights the main differences between DCTC and *SMESD*. Additionally, our results are indirectly compared to two approaches which are known to perform well in dynamic optimization, namely *RIGA-elit* and *dRepairRIGA* [19].

Table 11.2 Differences and similarities between DCTC and *SMESD*

	DCTC	*SMESD*
Search engine	Artificial Immune System based on T cells behavior	Multimembered Evolution Strategy
Population size	4	1
Number of mutation operators	3	1
Mutation rate	Fixed	It is decreased during the search process
Recombination operator	No	Yes
Constraint-handling mechanism	Discrimination between feasible and infeasible solutions. It uses the sum of constraint violations. No penalty function is required.	Discrimination between feasible and infeasible solutions. It uses the sum of constraint violations. No penalty function is required.
Extra diversity mechanism	No	It allows that the best infeasible solution which is closest to the boundary with the feasible region remaining into the population with some probability (given by the user).

The following experiments were performed for our proposed DCTC and for *SMESD* for validation purposes and in order to compare the performance of these two approaches. Both algorithms were implemented in C and the experiments were performed on a PC having an Intel Pentium P6000 processor, running at 1.87 GHz, and with 3 GB in RAM.

11.5.3.1 Benchmark Problems Setting

Table 11.3 indicates the parameters settings adopted for our proposed DCTC, for *SMESD*, for *RIGA-elit*, and for dRepairRIGA. The dynamic parameters were set as follows:

- Number of runs: 50
- Number of changes: 5/k

- Change frequency: 250, 500, and 1000 objective function evaluations per change
- Objective function severity of the changes (k): 0.25 (small), 0.5 (medium) and 1.0 (large). For G24_6a, G24_6c, and G24_6d only, k=1.0
- Constraints severity of the changes (S): 10 (small), 20 (medium) and 50 (large)

It is worth noting that optimal values (necessary to calculate the offline errors), for each period through functions, were not provided in [17]. Thus, we obtained them by executing DCTC and *SMESD* with a budget of 350000 objective function evaluations pro period (for each dynamic parameters setting). Then, we took the best solution for each period (choosing from the union of the solutions obtained by both DCTC and *SMESD*). Therefore, the offline errors for DCTC and *SMESD* were calculated using these optimal values. Since the comparison of the results of DCTC with respect to those of *RIGA-elit* and *dRepairRIGA* is indirect and, considering that in [19], the authors do not describe how the optimal values were found, we cannot guarantee that we used the same values adopted by them. Additionally, in [19], only the results for medium severity and when using 1000 objective function evaluations pro period are reported.

Table 11.3 Parameters settings for DCTC, *SMESD*, *RIGA-elit*, and *dRepairRIGA*

Parameter	DCTC	Parameter	*SMESD*	Parameter	*RIGA-elit*	Parameter	dRepairRIGA
VC	20	parents	10	popsize	25	popsize	20
CD4/ CD8	10	children	20	elitism	Yes	elitism	Yes
CM	5	Apply diversity mecha-nism	Yes	selection method	non-linear ranking	selection method	non-linear ranking
rep_{MC}	2	Selection ratio	0.97	mutation operator	uniform (P=0.15)	mutation operator	uniform (P=0.15)
$prob_{mut}$	0.9			crossover operator	arithmetic (P=0.1)	crossover operator	arithmetic (P=0.1)
clones	3			rand-inming	rate (P=0.3)	rand-inming	rate (P=0.3)
						reference popsize	5
						replace rate	0

In order to statistically determine if when we increase the change frequency, the objective function severity, the constraints severity or if when vary the dynamic features of the problems, our proposed DCTC produces results with significant differences, we performed an analysis of variance (ANOVA) taking into account the offline errors attained by our proposed DCTC from each run of all the experiments performed. Thus, the hypotheses considered were the following:

Null Hypothesis: there is no significant difference among the averages of the offline errors (*oe*). If there are differences, they are due to random effects.

Alternative Hypothesis: there is a combination of factor values for which the averages of the offline errors (*oe*) are significatively different and these differences are not due to random effects.

As the results (offline errors) do not follow a normal distribution, we applied the Kruskal-Wallis test, to perform the ANOVA and then the Turkey method in order to determine the experimental conditions for which significant differences exist. The results obtained by the ANOVA proved the Null Hypothesis for several combinations of parameters. However, the Alternative Hypothesis was proved, too. Tables 11.9 to 11.11 summarize the values of severity for which significant different results were detected.

11.6 Discussion of Results

The running time depends on the number of objective function evaluations and the test function itself. For instance, for G24_1, the running time taken by one run ranges from 9 to 200 milliseconds when 250 and 1000 objective function evaluations pro period are performed, respectively. For G24_3, the execution time taken by one run ranges from 12 to 243 milliseconds when 250 and 1000 objective function evaluations pro period are performed, respectively. Finally, for G24_3b, the running time taken by one run ranges from 12 to 239 milliseconds when 250 and 1000 objective function evaluations are performed, respectively.

Table 11.4 shows the results obtained for problems with both a dynamic objective function and dynamic constraints. If we fix the number of objective function evaluations pro period as well as the constraint severity values and increase the objective function severity values, we can see how for G24_3b, in general, the offline errors deteriorate. But there are significant differences only between the results produced when adopting low and medium values of k with respect to those obtained when k is large. Furthermore, when the constraint severity is large, the results which show significant differences are those produced with low values of k with respect to those obtained with medium and large values of k.

For G24_4, an increase in the objective function severity value gives rise to worse offline errors with our proposed DCTC. In this case, we obtain results with significant differences when the number of objective function evaluations pro period is equal to 250 and the constraint severity value is low. For 500 and 1000 evaluations, the results that show significant differences are those produced with low and medium values of k with respect to those obtained with large values of k.

For G24_5, an increase in the objective function severity value also deteriorates the offline errors produced by our proposed DCTC. In this case, we obtain significant differences when, in general, the constraint severity values are low and medium and k grows from low to medium. In general, when the constraint severity value is

Table 11.4 Offline errors (the standard deviation is shown in parentheses) for problems with dynamic objective function and dynamic constraints

			Dynamic Parameters								
				k=0.25			k=0.5			k=1.0	
Ev.	Probl.	Alg.	S=10	S=20	S=50	S=10	S=20	S=50	S=10	S=20	S=50
250	G24_3b	DCTC	**0.59** (0.15)	**0.54** (0.15)	**0.56** (0.14)	**0.55** (0.13)	**0.57** (0.13)	**1.15** (0.13)	1.67 (0.26)	**1.09** (0.29)	**1.68** (0.20)
		SMESD	0.85 (0.00)	0.85[1] (0.00)	0.87 (0.00)	0.78 (0.00)	0.83 (0.04)	1.44 (0.00)	**1.49** (0.25)	1.31 (0.14)	2.40 (0.00)
250	G24_4	DCTC	**0.43** (0.07)	**0.41** (0.05)	**0.28** (0.04)	**0.71** (0.08)	**0.62** (0.05)	**0.28** (0.06)	**1.53** (0.13)	**1.33** (0.09)	**1.33** (0.11)
		SMESD	0.82 (0.00)	0.80 (0.00)	0.69 (0.00)	1.05 (0.00)	0.99 (0.01)	0.68 (0.00)	1.91 (0.15)	2.17 (0.00)	2.14 (0.00)
250	G24_5	DCTC	**0.21** (0.02)	**0.18** (0.02)	**0.11** (0.02)	**0.34** (0.04)	**0.31** (0.04)	**0.12** (0.04)	0.46 (0.08)	0.28 (0.11)	**0.31** (0.09)
		SMESD	0.38 (0.01)	0.44 (0.11)	0.45 (0.03)	0.68 (0.10)	0.53 (0.03)	0.22 (0.00)	**0.43** (0.00)	**0.24** (0.01)	0.35 (0.00)
500	G24_3b	DCTC	**0.54** (0.16)	**0.49** (0.13)	**0.49** (0.10)	**0.49** (0.14)	**0.51** (0.11)	**1.03** (0.10)	**1.59** (0.23)	**0.93** (0.18)	**1.54** (0.16)
		SMESD	0.84 (0.00)	0.81 (0.00)	0.84[2] (0.01)	0.74 (0.00)	0.81 (0.00)	1.40 (0.01)	1.72 (0.00)	1.79 (0.06)	2.36 (0.00)
500	G24_4	DCTC	**0.36** (0.05)	**0.35** (0.02)	**0.23** (0.03)	**0.63** (0.05)	**0.55** (0.04)	**0.20** (0.04)	**1.41** (0.06)	**1.26** (0.08)	**1.20** (0.07)
		SMESD	0.81 (0.00)	0.81[3] (0.02)	0.67 (0.00)	0.83[4] (0.04)	0.94 (0.00)	0.61 (0.00)	2.23 (0.00)	2.05 (0.00)	2.03 (0.01)
500	G24_5	DCTC	**0.18** (0.01)	**0.15** (0.01)	**0.07** (0.01)	**0.28** (0.02)	**0.26** (0.03)	**0.07** (0.03)	0.38 (0.04)	**0.20** (0.05)	**0.25** (0.07)
		SMESD	0.47 (0.08)	0.58 (0.19)	0.52 (0.16)	0.47 (0.02)	0.45 (0.03)	0.17 (0.00)	**0.32** (0.00)	0.37 (0.28)	0.73 (0.00)
1000	G24_3b	DCTC	**0.47** (0.12)	**0.43** (0.08)	**0.39** (0.06)	**0.41** (0.12)	**0.45** (0.09)	**0.98** (0.09)	**1.44** (0.18)	**0.83** (0.14)	**1.45** (0.08)
		SMESD	0.87[5] (0.03)	0.82[6] (0.02)	0.82 (0.03)	0.73 (0.00)	0.81 (0.00)	1.49[7] (0.09)	2.23 (0.00)	1.65 (0.01)	2.00 (0.29)
1000	G24_4	DCTC	**0.32** (0.02)	**0.32** (0.02)	**0.19** (0.01)	**0.57** (0.02)	**0.50** (0.02)	**0.15** (0.02)	**1.36** (0.05)	**1.17** (0.05)	**1.13** (0.05)
		SMESD	0.81 (0.00)	0.80[8] (0.02)	0.68[9] (0.02)	1.06[10] (0.06)	0.87[11] (0.09)	0.61 (0.00)	2.07 (0.22)	1.57 (0.22)	2.01 (0.00)
1000	G24_5	DCTC	**0.17** (0.02)	**0.12** (0.02)	**0.06** (0.01)	**0.25** (0.01)	**0.23** (0.01)	**0.03** (0.02)	**0.34** (0.03)	**0.15** (0.03)	**0.20** (0.03)
		SMESD	0.56 (0.15)	0.73 (0.07)	1.00 (0.17)	0.57 (0.30)	0.72 (0.24)	0.83 (0.05)	0.78 (0.12)	0.64 (0.00)	0.72 (0.00)

[1] rf=32.0 - [2] rf=68.0 - [3] rf=44.0 - [4] rf=6.0 - [5] rf=30.0 - [6] rf=70.0 - [7] rf=46.0 - [8] rf=80.0 - [9] rf=60.0- [10] rf=54.0 - [11] rf=68.0

large, the results show significant differences only between those obtained with low and medium values k and those corresponding to large values of k.

On the other hand, if we fix the number of objective function evaluations pro period as well as the objective function severity values and we increase the constraint severity value, we can see how G24_3b offline errors with low k improve but without significant differences. If we use medium values of k, the results get worse but show significant differences only between the results produced with low and medium values of S with respect to those obtained when S is large. In general, with large values of k, the best results are obtained with medium values of S showing significant differences with respect to the results obtained with low and medium values of S.

For G24_4, if we adopt either a low or a large value of k, an increase of S improves the results but not with significant differences. With a medium value of k, the results also improve, showing significant differences only between the results produced with low and medium values of S with respect to those obtained when S is large.

For G24_5, an increase in S improves the results produced by our proposed DCTC. For low values of k, the results show significant differences only between the results produced with either a low or a large S. For medium values of k, the results show significant differences between the results produced with low and medium values of S with respect to those obtained with large values of S. For large values of k, the best results are obtained with medium values of S, showing significant differences only with respect to those obtained with low values of S, but not with respect to those produced with large values of S.

An increase in the number of objective function evaluations per change does not produce results with significant differences for G24_3b and G24_4. For G24_5, the results obtained for 250 evaluations present significant differences with respect to those found for 1000 evaluations with a large value of k and either a low or a medium S.

Our proposed DCTC always outperforms *SMESD*, when compared on problems with dynamic objective function and dynamic constraints, except for four cases as shown in Table 11.4. Furthermore, in one case, (the eleventh experiment), *SMESD* fails to find feasible solutions in all changes for all runs, while our proposed DCTC had success in the same task.

Tables 11.5 and 11.6 show the results obtained for problems with a dynamic objective function and fixed constraints. If we fix the number of objective function evaluations pro period and we increase the objective function severity value, we can see how for G24_1 and G24_2 the offline errors get worse. But the results show significant differences only between the results produced when using low and large values of k, for 250 and 500 evaluations per change.

For G24_6a, G24_6c, and G24_8b, in general, offline errors get worse when k grows. But the results show significant differences only when they are produced with low values of k with respect to those produced with medium and large values of k.

For G24_6d, an increase in the objective function severity value deteriorates the offline errors but not with significant differences.

An increase in the number of objective function evaluations pro period when k is fixed produces better results with significant differences.

Table 11.5 Offline errors (the standard deviation is shown in parentheses) for problems with dynamic objective function and fixed constraints

Ev.	Probl.	Alg.	Dynamic Parameters k=0.25	k=0.5	k=1.0
250	G24_1	DCTC	**0.03** (0.01)	**0.05** (0.03)	**0.12** (0.04)
		SMESD	1.66 (0.00)	1.58 (0.00)	2.39 (0.03)
250	G24_2	DCTC	**0.08** (0.03)	**0.12** (0.03)	0.18 (0.10)
		SMESD	0.77 (0.01)	0.38 (0.00)	**0.16** (0.00)
250	G24_8b	DCTC	**0.11** (0.04)	**0.25** (0.08)	**0.58** (0.19)
		SMESD	0.55 (0.01)	0.74 (0.00)	0.76 (0.00)
500	G24_1	DCTC	**0.00** (0.00)	**0.01** (0.01)	**0.03** (0.02)
		SMESD	1.65 (0.00)	1.58 (0.01)	1.74 (0.00)
500	G24_2	DCTC	**0.05** (0.02)	**0.06** (0.03)	**0.12** (0.07)
		SMESD	0.84 (0.08)	0.26 (0.07)	0.57 (0.00)
500	G24_8b	DCTC	**0.04** (0.02)	**0.12** (0.06)	**0.29** (0.13)
		SMESD	0.51 (0.00)	0.69 (0.09)	1.07 (0.00)
1000	G24_1	DCTC	**0.00** (0.00)	**0.00** (0.00)	**0.00** (0.00)
		SMESD	1.65 (0.00)	1.57 (0.00)	2.32 (0.00)
1000	G24_2	DCTC	**0.03** (0.01)	**0.03** (0.02)	**0.04** (0.04)
		SMESD	1.31 (0.12)	0.79 (0.12)	0.49 (0.20)
1000	G24_8b	DCTC	**0.01** (0.01)	**0.03** (0.02)	**0.07** (0.07)
		SMESD	0.51 (0.00)	0.72 (0.01)	1.03 (0.00)

Table 11.6 Offline errors (the standard deviation is shown in parentheses) for problems with dynamic objective function and fixed constraints

Ev.	Probl.	Algorithms DCTC	*SMESD*
250	G24_6a	**0.26** (0.38)	1.76 (0.00)
250	G24_6c	0.12 (0.05)	**0.11** (0.00)
250	G24_6d	**0.14** (0.18)	0.55 (0.00)
500	G24_6a	**0.06** (0.12)	1.75 (0.00)
500	G24_6c	**0.06** (0.03)	0.10 (0.04)
500	G24_6d	**0.04** (0.14)	0.50 (0.00)
1000	G24_6a	**0.02** (0.02)	1.75 (0.00)
1000	G24_6c	**0.04** (0.03)	0.13 (0.00)
1000	G24_6d	**0.00** (0.00)	0.13 (0.00)

Our proposed DCTC always outperforms *SMESD*, when compared on problems with dynamic objective function and static constraints, except for one case, as shown in Table 11.5. In these problems both approaches found, for all runs, feasible solutions, for all changes.

Table 11.7 shows the results obtained for problems with a static objective function and dynamic constraints. If we fix the number of objective function evaluations pro period and increase the constraint severity value we can see how, for G24_3, the offline errors improve. In general, the results show significant differences only between the results produced with low values of S with respect to those obtained with medium and large values of S.

Table 11.7 Offline errors (the standard deviation is shown in parentheses) for problems with static objective function and dynamic constraints

Ev.	Probl.	Alg.	Dynamic Parameters S=10	S=20	S=50
250	G24_3	DCTC	0.16 (0.15)	0.15 (0.21)	0.12 (0.07)
		SMESD	**0.12** (0.00)	**0.04** (0.01)	**0.01** (0.00)
250	G24_7	DCTC	0.15 (0.02)	0.11 (0.03)	0.10 (0.03)
		SMESD	**0.14** (0.00)	**0.03** (0.01)	**0.08** (0.05)
500	G24_3	DCTC	0.13 (0.14)	0.10 (0.13)	0.10 (0.11)
		SMESD	**0.10**[1] (0.00)	**0.02** (0.00)	**0.00** (0.00)
500	G24_7	DCTC	**0.12** (0.02)	0.07 (0.02)	0.06 (0.02)
		SMESD	**0.12** (0.01)	**0.04** (0.03)	**0.00** (0.00)
1000	G24_3	DCTC	0.11 (0.03)	0.05 (0.03)	0.05 (0.04)
		SMESD	**0.09** (0.01)	**0.02** (0.00)	**0.00** (0.00)
1000	G24_7	DCTC	**0.10** (0.02)	0.05 (0.01)	0.04 (0.01)
		SMESD	0.11 (0.01)	**0.02** (0.00)	**0.00** (0.00)

[1] rf=44.0

Finally, an increase in the number of objective function evaluations per change produces results with significant differences for G24_3 and medium values of S as well as for a number of evaluations of 250 and 1000. For G24_7, the results that present significant differences are those found for 250 and 1000 evaluations with low values of S, as well as the results produced with 250 evaluations with respect to those obtained with 500 and 1000 evaluations, using a medium value of S. For those two problems, the results show significant differences when S is large.

SMESD outperforms our proposed DCTC in all problems with static objective function and dynamic constraints, except for one case and, in another case (see Table 11.7), *SMESD* fails to find feasible solutions in all changes for some runs, whereas our proposed DCTC found feasible solutions for all changes in all the runs performed.

Table 11.8 shows the results for DCTC vs *RIGA-elit* and dRepairRIGA. Here we can note that *RIGA-elit* outperforms DCTC only in one test case while DCTC is superior to dRepairRIGA in seven of the eleven test cases adopted.

Table 11.8 Offline errors (the standard deviation is shown in parentheses) for DCTC vs *RIGA-elit* and dRepairRIGA

Probl.	Algorithms DCTC	*RIGA-elit*	*dRepairRIGA*
G24_1	**0.00** (0.00)	0.40 (0.04)	0.08 (0.01)
G24_2	**0.03** (0.02)	0.28 (0.02)	0.16 (0.02)
G24_3	0.05 (0.03)	0.34 (0.04)	**0.02** (0.00)
G24_3b	0.45 (0.09)	0.47 (0.05)	**0.05** (0.00)
G24_4	0.50 (0.02)	0.49 (0.07)	**0.14** (0.02)
G24_5	0.23 (0.01)	0.25 (0.03)	**0.15** (0.01)
G24_6a	**0.02** (0.02)	0.45 (0.05)	0.36 (0.03)
G24_6c	**0.04** (0.03)	0.41 (0.04)	0.32 (0.03)
G24_6d	**0.00** (0.00)	0.42 (0.02)	0.31 (0.02)
G24_7	**0.05** (0.01)	0.45 (0.05)	0.15 (0.03)
G24_8b	**0.03** (0.02)	1.08 (0.11)	0.34 (0.05)

In order to determine if DCTC is able to recover and converge to a solution immediately after a change, we analyze the plot of *RR*/*ARR* scores displayed in Figures 11.1, 11.2, 11.3, 11.4, and 11.5.

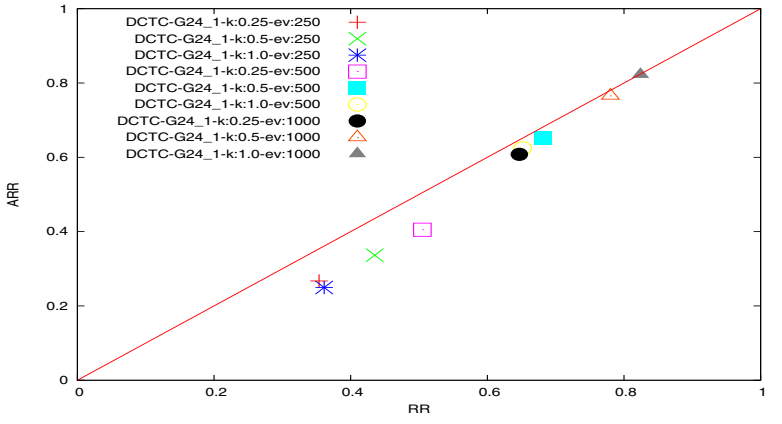
For G24_1 and G24_2 (see Figures 11.1 (a) and (b)), our proposed DCTC found, on the median run, solutions close to the optimum. As the number of objective function evaluations grows, the algorithm recovers faster and gets closer to the new optimum. Also, objective function severity has a negative impact on convergence when it is increased.

For those problems in which only the constraints change (see Figures 11.1 (c) and 11.3), the algorithm found solutions close to the optimum when the constraint severity was larger. When constraint severity was low, the algorithm normally converged to local optima.

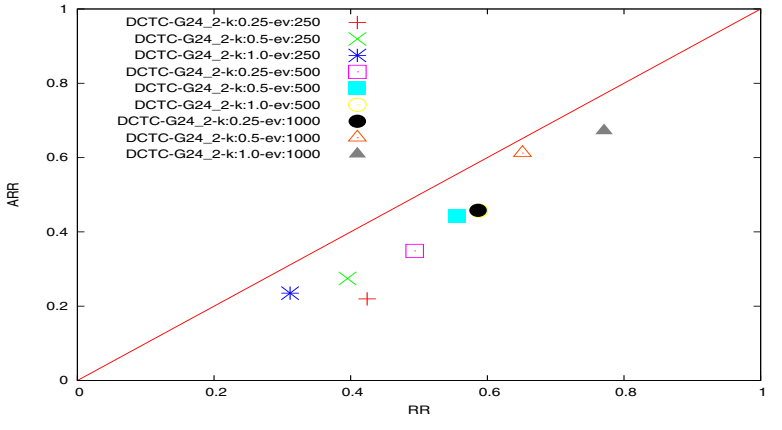
For G24_6a, G24_6c, and G24_6d with 500 and 1000 evaluations per change (see Figure 11.2 (a)) the algorithm had a perfect and an almost perfect performance regarding convergence behavior and recovery speed. But, with 250 evaluations per change it presents moderate convergence behavior and recovery speed.

For G24_3b and G24_4, with 250 evaluations per change (see Figures 11.3 (a) and 11.4 (a)), our proposed DCTC presented relatively moderate convergence behavior and recovery speed.

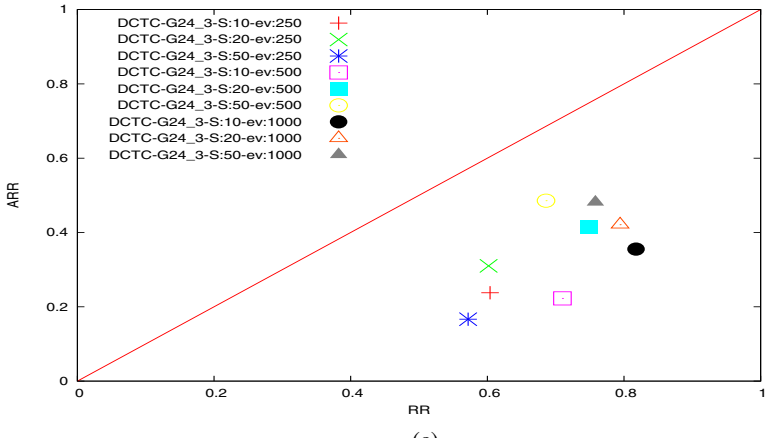
For G24_3b with 500 and 1000 evaluations per change and G24_5, with 250 and 500 evaluations per change (see Figures 11.3 (b) and (c) and Figures 11.5 (b) and (c)), our proposed DCTC presented good convergence behavior and recovery speed.



(a)

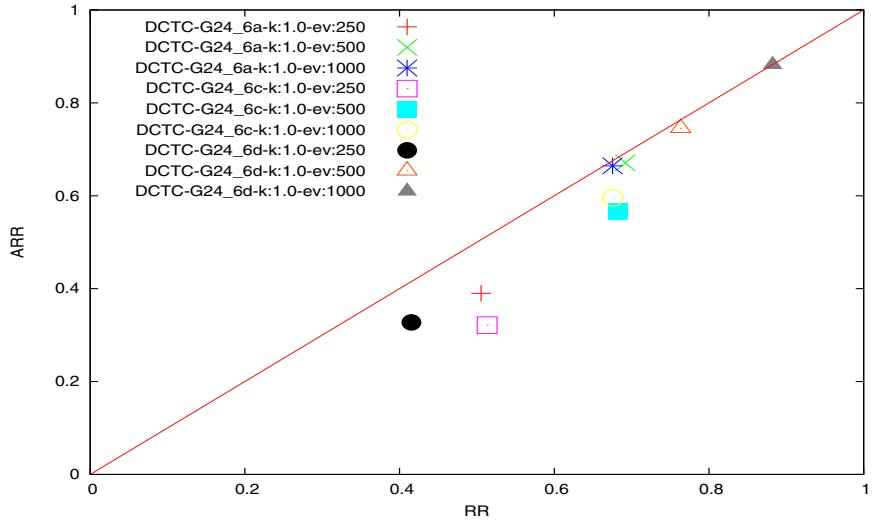


(b)

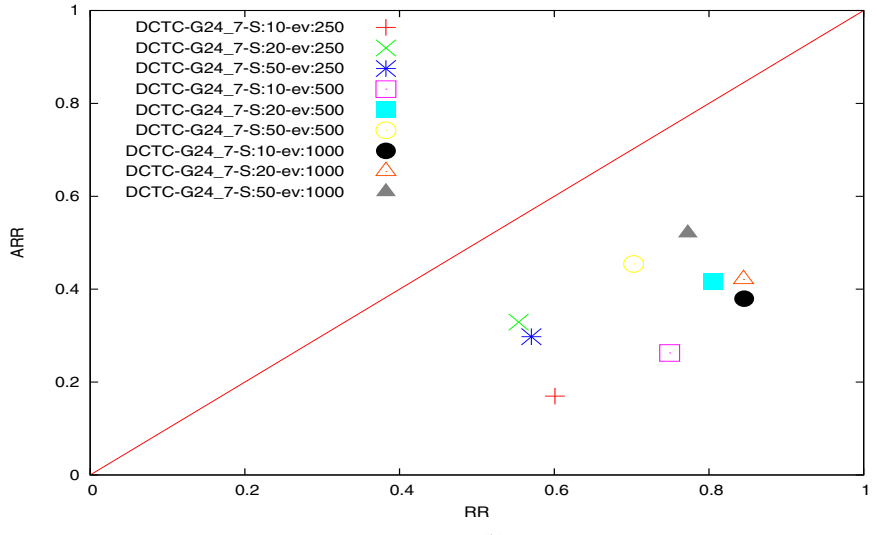


(c)

Fig. 11.1 RR/ARR for G24_1, G24_2 and G24_3

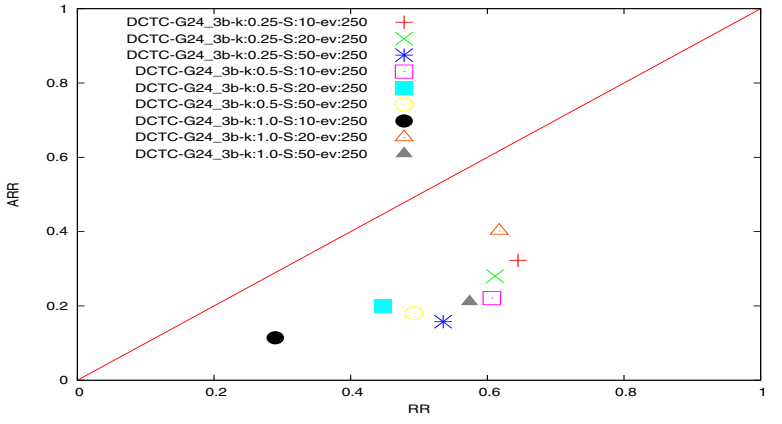


(a)

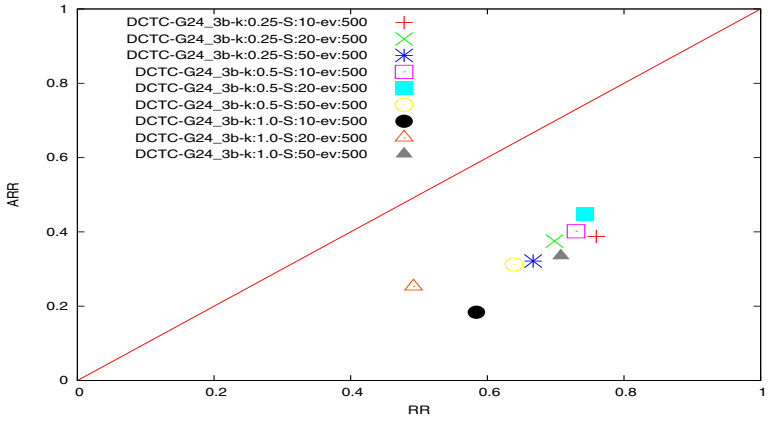


(b)

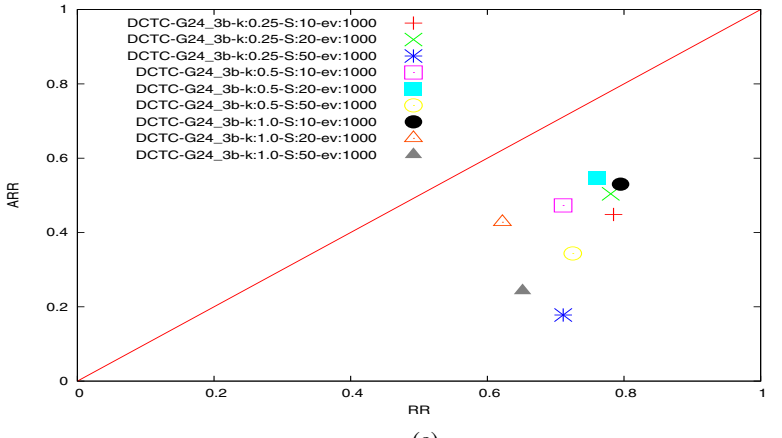
Fig. 11.2 RR/ARR for G24_6a and G24_7



(a)

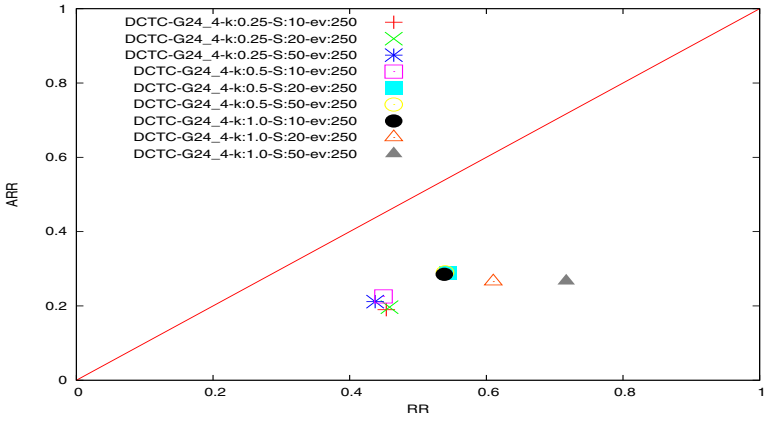


(b)

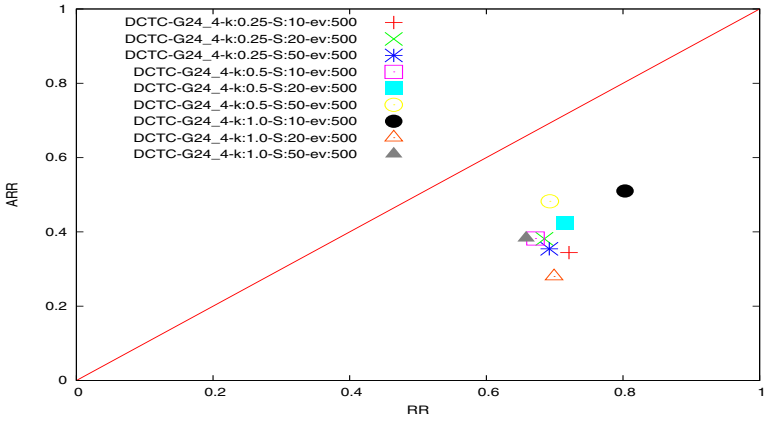


(c)

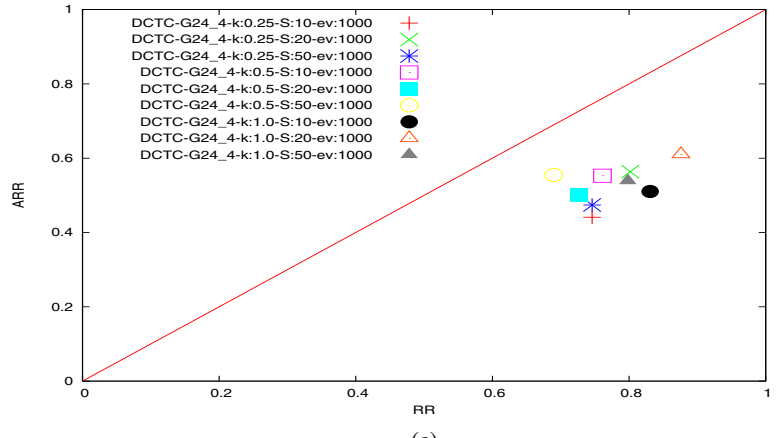
Fig. 11.3 RR/ARR for G24_3b



(a)

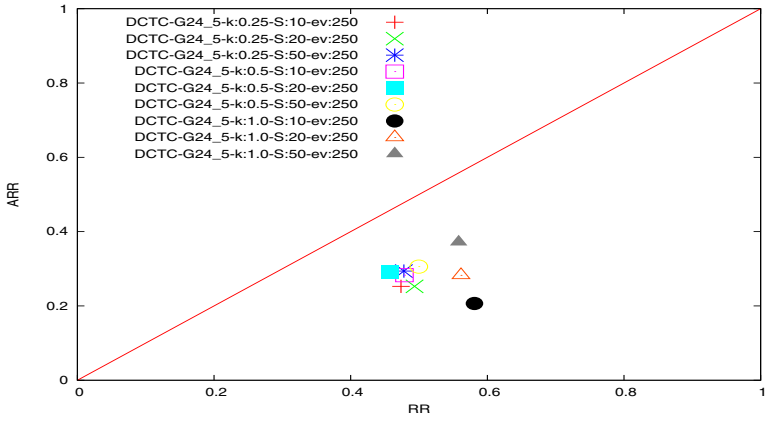


(b)

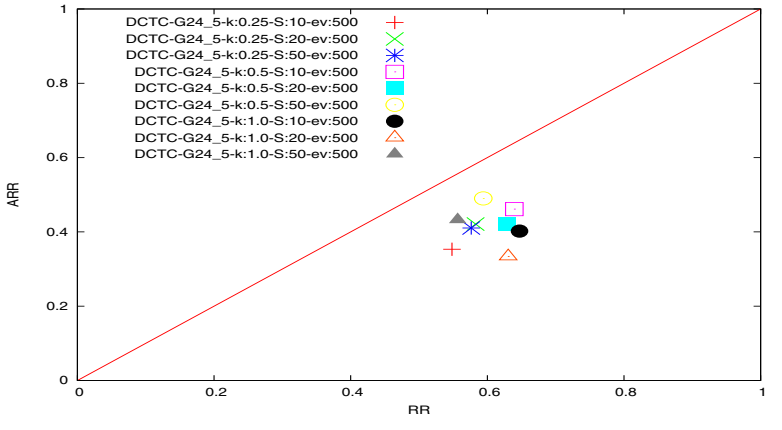


(c)

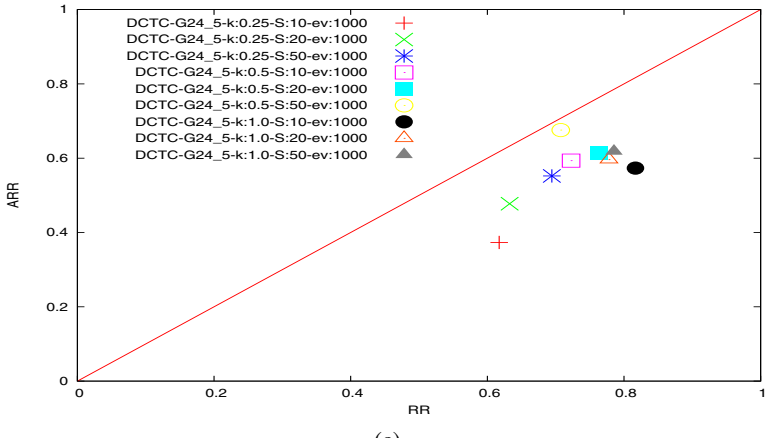
Fig. 11.4 RR/ARR for G24_4



(a)



(b)



(c)

Fig. 11.5 RR/ARR for G24_5

Particularly, for G24_5, with 250 evaluations per change, the larger the objective function severity, the better becomes the convergence behavior.

For G24_4 with 500 evaluations per change (see Figure 11.4 (b)), in general, the algorithm presented a fast recovery speed but the solutions found were not very close to the optimum.

For G24_4 and G24_5, with 1000 evaluations per change (see Figures 11.4 (c) and 11.4 (c)), the convergence behavior of our proposed DCTC is very good and the recovery speed is high.

In order to compare the effect that different features of the dynamic constrained problems had on performance, Nguyen et al. [17] proposed to contrast the offline errors produced over pairs of problems. In this work, the following comparisons were made:

- Static constraints versus dynamic constraints - G24_1 vs G24_4 and G24_2 vs G24_5.
- Moving constraints that do not expose better optima versus moving constraints that expose better optima - G24_3 vs G24_3b.
- Connected feasible regions versus disconnected feasible regions - G24_6c vs G24_6d.
- Optima in the constraints boundary versus optima that are not in the constraints boundary - G24_4 vs G24_5.

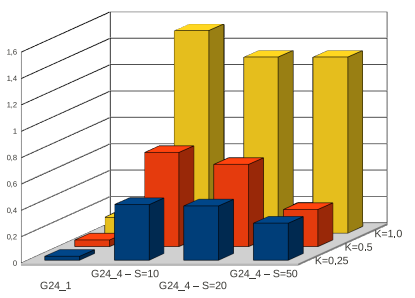
The offline errors produced by our proposed DCTC for problems with static constraints and dynamic constraints (see Figures 11.6 (a) to (c)) clearly show the negative impact on performance when constraints change over time. A statistical analysis of variance indicates that the offline errors obtained for G24_1 have significant differences with respect to the offline errors obtained for G24_4. Note that for G24_4, the larger the constraint severity, the better the performance.

For G24_2 versus G24_5, the offline errors also deteriorate when the problem changes its constraints over time (see Figures 11.6 (d) to (f)), but the statistical analysis of variance indicates that the offline errors obtained for G24_5, with a medium value of S, are not significantly different from the offline errors obtained for G24_2, regardless of the number of evaluations between changes.

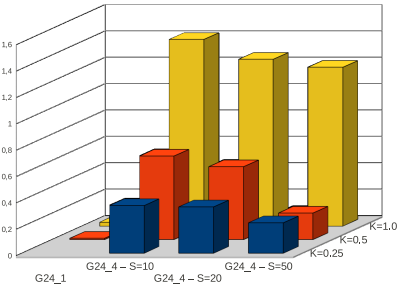
The algorithm had better performance when the optimum was in the constraint boundary than when it was not (see Figure 11.7 (a)), showing only significant differences with large values of k and a few evaluations per change. The opposite situation occurs when we compare the results obtained for G24_4 and G24_5 (see Figures 11.7 (b) to (d)). Also, they always presented significant differences.

When the algorithm had only a few evaluations to perform per state (or change), moving inside the connected feasible regions was easier than moving between disconnected feasible regions. But, if we could perform more evaluations per change, moving between disconnected regions became easier (see Figure 11.8 (a)), showing significant differences in the offline errors for all the tested cases.

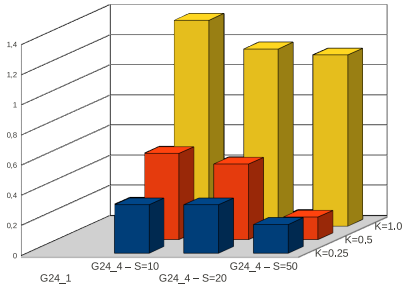
The exposition of better optima when the constraints change, had a negative impact on the performance of our proposed DCTC, showing significant differences for all the tested cases (see Figures 11.8 (b) to (d)).



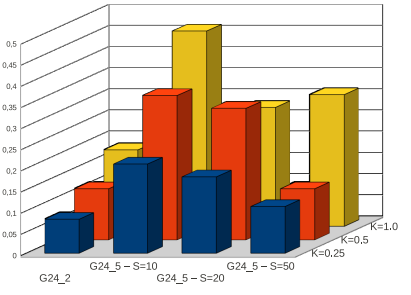
a - ev=250



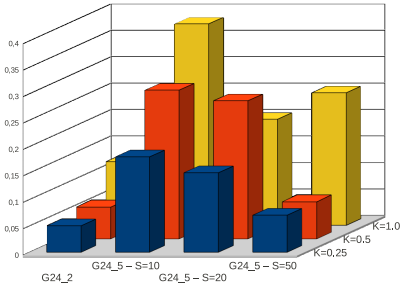
b - ev=500



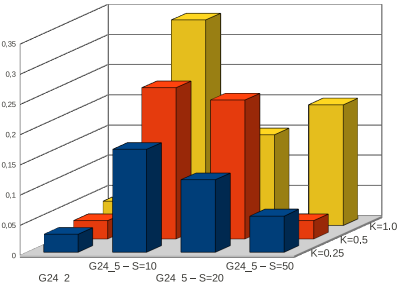
c - ev=1000



d - ev=250



e - ev=500



f - ev=1000

Fig. 11.6 The effect of two different problem features on the performance of DCTC. G24_1 versus G24_4 and G24_2 versus G24_5= Static constraints versus dynamic constraints. Performance is evaluated based on the offline error

Table 11.9 Summary of the ANOVA results. - indicates that significant differences were detected

Probl.	Eval.	Pairs of severity values for the results having significant differences
G24_1	250	low k and high k
	500	low k and high k
	1000	-
G24_2	250	low k and high k
	500	low k and high k / medium k and high k
	1000	-
G24_3	250	low S and medium S
	500	low S and high S / low S and medium S
	1000	low S and high S / low S and medium S
G24_6a	250	low k and medium k / low k and high k
	500	low k and medium k
	1000	low k and medium k / low k and high k
G24_6c	250	low k and medium k / low k and high k
	500	low k and medium k / low k and high k
	1000	low k and medium k / low k and high k
G24_6d	250	-
	500	-
	1000	-
G24_7	250	low S and high S / low S and medium S
	500	low S and high S / low S and medium S
	1000	low S and high S / low S and medium S
G24_8b	250	low k and medium k / low k and high k / medium k and high k
	500	low k and medium k / low k and high k / medium k and high k
	1000	low k and high k

Table 11.10 Summary of ANOVA results. - indicates that significant differences were detected

Probl.	Eval.	k	Pairs of severity values for the results having significant differences
G24_3b	250 / 500 / 1000	small	-
	250/ 500 / 1000	medium	low S and high S / medium S and high S
	250 / 500 / 1000	large	low S and medium S / medium S and high S
G24_4	250 / 500 / 1000	small	-
	250 / 500 / 1000	medium	low S and high S / medium S and high S
	250 / 500 / 1000	large	-
G24_5	250 / 500 / 1000	small	low S and high S
	250 / 500 / 1000	medium	low S and high S / medium S and high S
	250 / 500 / 1000	large	low S and medium S / low S and high S

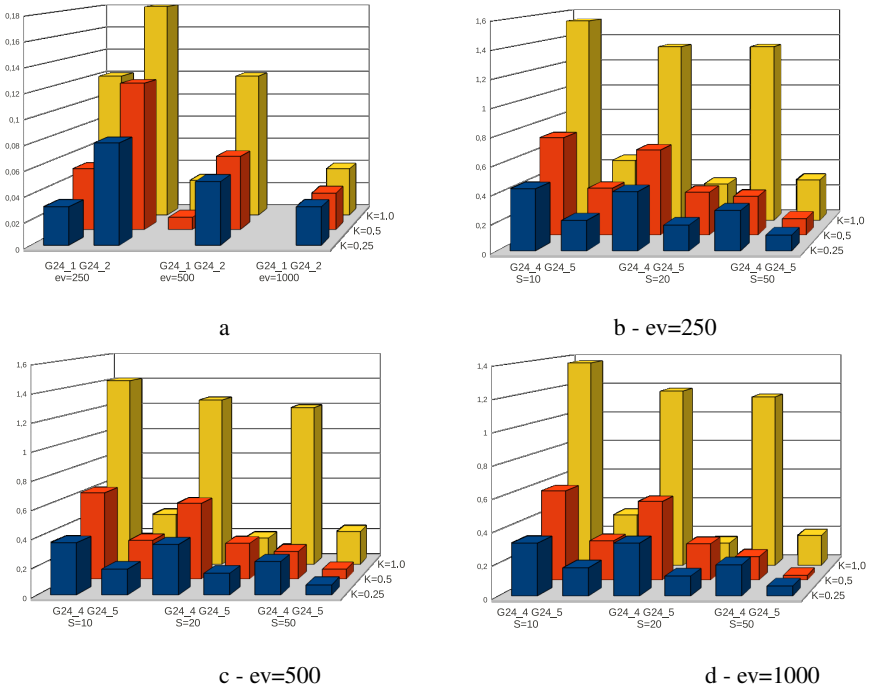


Fig. 11.7 The effect of two different problem features on the performance of DCTC. G24_1 versus G24_2 and G24_4 versus G24_5= Optimum in the constraint boundary versus Optimum not in the constraint boundary. Performance was evaluated based on the offline error

11.6.1 Increasing the Number of Changes per Run

In order to determine if the performance of DCTC and *SMESD* gets affected when more than five changes occur, in the case in which severity is high (k= 0.01 and S= 50) and the minimum time pro period is granted (only 250 objective function evaluations), we ran these two algorithms allowing fifty changes (in 50 independent runs) for each test problem. The results of these experiments are shown in Table 11.12.

First, it can be seen that DCTC outperforms *SMESD* in all the test cases, except for G24_7 but here, *SMESD* could not find feasible solutions for every period ($rf = 62.0$). It is worth noting that, for G24_3 and G24_3b, even when the offline errors of *SMESD* are zero, their rf values are zero, as well. This means that *SMESD* could not find, in any run, a feasible solution for each period while DCTC could do it.

On the other hand, when we consider the results obtained by DCTC for 5 against 50 changes (with the highest severity), in general (8 from 11 cases) the offline errors improved when more changes were allowed, while, regarding *SMESD*, on 6 of the

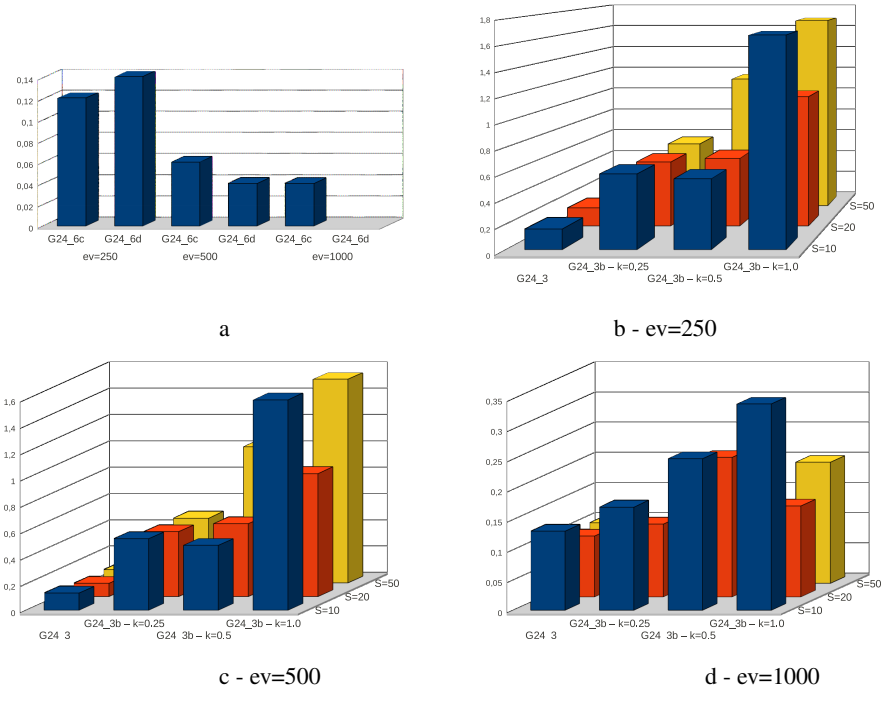


Fig. 11.8 The effect of four different problem features on the performance of DCTC. G24_6c versus G24_6d = Connected feasible regions versus disconnected feasible regions and G24_3 versus G24_3b= Moving constraints do not expose a better optimum versus moving constraints expose a better optimum. Performance was evaluated based on the offline error

11 test cases the results were worst. Thus, we can think that *SMESD* loses its ability to react to changes when these are increased, while DCTC properly maintains diversity during the search process.

11.7 Conclusions

In this chapter, we have analyzed the behavior of an adaptive immune system called Dynamic Constrained T-Cell (DCTC) for solving dynamic constrained optimization problems. One of the strengths we can highlight about DCTC is the few number of parameters that it requires. Furthermore, and analogously to other techniques that do not rely on a penalty function to handle constraints, DCTC does not need to define a penalty factor, which normally has to take a specific value for each problem at hand. An adaptation of an existing algorithm, which was originally used to solve static constrained optimization problems, was used to compare the results obtained by our proposed DCTC on eleven constrained optimization problems which present

Table 11.11 Summary of ANOVA results for G24_1 vs G24_4, G24_2 vs G24_5, G24_4 vs G24_5, G24_1 vs G24_2 and G24_6c vs G24_6d. - indicates that significant differences were detected

First function vs Second Function	Eval.	Did the results obtained for the first function have significant differences with respect to the results obtained for the second function with any dynamic parameter?
G24_1 vs G24_4	250	Always
	500	Always
	1000	Always
G24_2 vs G24_5	250	Yes, when S is small and when S is medium
	500	Yes, when S is small and when S is medium
	1000	Always
G24_4 vs G24_5	250	Always
	500	Always
	1000	Always
G24_3 vs G24_3b	250	Always
	500	Always
	1000	Always
G24_1 vs G24_2	250	Yes, when k is small and when k is medium
	500	Always
	1000	Always
G24_6c vs G24_6d	250	Always
	500	Always
	1000	Always

Table 11.12 Offline errors (the standard deviation is shown in parentheses) for dynamic constrained problems performing 50 changes

Probl.	Algorithms DCTC	*SMESD*
G24_1	**0.06** (0.02)	2.32 (0.00)
G24_2	**0.20** (0.04)	0.55 (0.04)
G24_3	**0.19** (0.05)	0.00 (0.00)[1]
G24_3b	**0.49** (0.17)	0.86 (0.01)[2]
G24_4	**0.16** (0.04)	1.03 (0.07)[3]
G24_5	**0.11** (0.02)	0.32 (0.03)
G24_6a	**0.08** (0.04)	1.80 (0.02)
G24_6c	**0.09** (0.02)	0.23 (0.00)
G24_6d	**0.08** (0.06)	0.97 (0.00)
G24_7	0.05 (0.01)	**0.00** (0.00)[4]
G24_8b	**0.20** (0.07)	0.64 (0.13)

[1] *rf=0.0* - [2] *rf=0.0* - [3] *rf=54.0* - [4] *rf=62.0*

several forms of dynamism (both in the objective function and in the constraints). Additionally, DCTC was also indirectly compared to two approaches used to solve dynamic constrained optimization problems: *RIGA-elit* and dRepairRIGA.

For problems with a dynamic objective function and dynamic constraints, an increase in the objective function severity produces a poorer performance of our proposed DCTC. However, in general, the results that show significant differences are those found with low severity with respect to those found with large severity. An increase in the constraints severity improves the offline errors, showing, in some cases, significant differences, generally between the results found with low severity with respect to those found with a large severity. In general, an increase in the number of evaluations per change improves the offline errors but not with significant differences.

For those problems in which the objective function is dynamic and the constraints are static, in general, an increase in the severity has a negative impact on the behavior of our proposed DCTC. In this case, there are significant differences between the results obtained with low severity with respect to those obtained with a large severity. In this type of problems, an increase in the number of evaluations per change improves the offline errors with significant differences.

In problems in which the objective function is static and the constraints change over time, an increase in the severity improves the results. In this case, there are significant differences when using a low severity with respect to the use of a medium and a large severity. An increase in the number of evaluations per change improves the offline errors but significant differences are detected on results found with low severity, with respect to those obtained with a large severity. Regarding the poor behavior of DCTC in problems that present dynamic constraints, our hypothesis is the following. When the constraint severity is low, it is likely that many of the feasible solutions found so far keep their feasibility. However, in this case, the algorithm has converged to a local optimum and it remains trapped there. On the other hand, when the constraint severity is large, feasible solutions will become infeasible and viceversa. This causes the search to be redirected to the new feasible regions. This situation can be observed in both DCTC and *SMESD*.

When a global optimum switches between disconnected feasible regions and the constraints change, for our proposed DCTC it is more difficult to solve the problem than when the constraints are static.

For all the test problems adopted, we could see, in the median run, that a larger number of objective function evaluations allowed us to keep improving the solutions in that period.

When the problem presented a dynamic objective function and dynamic constraints and the number of objective function evaluations per change was low, the results obtained were not very good. But, we could see that if we increased this number, the results improved, showing significant differences in some cases. This leads us to believe that our proposed approach is able to adapt well to dynamic environments but requires a minimum number of evaluations in order to reach some stability.

Our proposed DCTC was found to be superior to *SMESD* in problems with a dynamic objective function and dynamic constraints and in problems with a dynamic objective function and static constraints. There were only five cases in which *SMESD* outperformed our proposed DCTC when using such types of problems. On the other hand, *SMESD* showed a better behavior than our proposed DCTC in problems having a static objective function and dynamic constraints. When we compared our results against those of *RIGA-elit*, we could note the superior performance of DCTC in all but one test case. On the other hand, DCTC showed to be competitive with respect to *dRepairRIGA*, overcoming it in seven of the eleven test cases adopted. As part of our future work, we aim to improve the mechanisms to maintain diversity of our approach, mainly when dealing with problems in which a change in the constraints gives rise to a new optimum. It is also desirable to improve the exploratory capabilities of our proposed algorithm so that it can be more effective in the test problems in which it was outperformed by *SMESD*. Thus, taking into account the performed experiments and the results obtained from them, we can suggest that DCTC should be suitable for solving problems having a dynamic objective function and either static or dynamic constraints. However, our aim is to improve the behavior of our proposed DCTC in problems having a static objective function and dynamic constraints. Finally, we would also like to extend our approach for solving multi-objective dynamic constrained optimization problems.

Acknowledgments. The first two authors acknowledge support from the Universidad Nacional de San Luis, the Agencia Nacional para promover la Ciencia y Tecnología (ANPCYT) and project no. PROICO 317902. The third author acknowledges support from CONACyT project no. 103570 and from the UMI LAFMIA 3175 CNRS at CINVESTAV-IPN.

References

[1] Aragón, V., Esquivel, S., Coello Coello, C.: Optimizing Constrained Problems through a T-Cell Artificial Immune System. Journal of Computer Science & Technology 8(3), 158–165 (2008)

[2] Aragón, V., Esquivel, S., Coello Coello, C.: Solving constrained optimization using a t-cell artificial immune system. Revista Iberoamericana de Inteligencia Artificial 12(40), 7–22 (2008)

[3] Aragón, V., Esquivel, S., Coello Coello, C.: Artificial Immune System for Solving Global Optimization Problems. Revista Iberoamericana de Inteligencia Artificial (AEPIA) 14(46), 3–16 (2010) ISSN: 1137-3601

[4] Aragón, V., Esquivel, S., Coello Coello, C.: A Modified Version of a T-Cell Algorithm for Constrained Optimization Problems. International Journal for Numerical Methods in Engineering 84(3), 351–378 (2010)

[5] Aragón, V.: Optimización de Problemas con Restricciones a través de Heurísticas BioInspiradas. PhD Tesis

[6] Branke, J.: Evolutionary Optimization in Dynamic Environments. Kluwer Academic Publishers (2002)

[7] Bretscher, P., Cohn, M.: A theory of self-nonself discrimination. Science 169, 1042–1049 (1970)

[8] Dasgupta, D., Nino, F.: Immunological Computation: Theory and Applications. Auerbach Publications, Boston (2008)
[9] Deb, K., Udaya Bhaskara Rao, N., 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)
[10] Jula, H., Dessouky, M., Ioannou, P., Chassiakos, A.: Container movement by trucks in metropolitan networks: modeling and optimization. Transportation Research Part E 41, 235–259 (2005)
[11] Mailler, R.: Comparing two approaches to dynamic, distributed constraint satisfaction. In: Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1049–1056. ACM, New York (2005), doi:10.1145/1082473.1082632
[12] Male, D., Brostoff, J., Roth, D., Roitt, I.: Inmunology. Mosby, 7th edn. (2006)
[13] Matzinger, P.: Tolerance, danger and the extend family. Annual Review of Immunology 12, 991–1045 (1994)
[14] Mertens, K., Holvoet, T., Berbers, Y.: The DynCOAA algorithm for dynamic constraint optimization problems. In: Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), pp. 1421–1423. ACM, New York (2006), doi:10.1145/1160633.1160898
[15] Mezura Montes, E., Coello Coello, C.: A Simple Multi-Membered Evolution Strategy to Solve Constrained Optimization Problems. IEEE Transactions on Evolutionary Computation 9(1), 1–17 (2005)
[16] Modi, P.J., Jung, H., Tambe, M., Shen, W.-m., Kulkarni, S.: A Dynamic Distributed Constraint Satisfaction Approach to Resource Allocation. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 685–700. Springer, Heidelberg (2001)
[17] Nguyen, T., Yao, X.: Continuous Dynamic Constrained Optimisation - The Challenges. IEEE Transactions on Evolutionary Computation, 321–354 (2010)
[18] Nguyen, T., Yao, X.: Solving dynamic constrained optimisation problems using repair methods (2011)
[19] Richter, H.: A study of dynamic severity in chaotic fitness landscapes. The 2005 IEEE Congress on Evolutionary Computation 3, 2824–2831 (2005)
[20] Richter, H., Yang, S.: Memory Based on Abstraction for Dynamic Fitness Functions. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., McCormack, J., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Uyar, A.Ş., Yang, S. (eds.) EvoWorkshops 2008. LNCS, vol. 4974, pp. 596–605. Springer, Heidelberg (2008)
[21] Richter, H., Yang, S.: Learning in Abstract Memory Schemes for Dynamic Optimization. In: Proceedings of the 2008 Fourth International Conference on Natural Computation, vol. 1, pp. 86–91. IEEE Computer Society, Washington, DC (2008)
[22] Richter, H.: Detecting change in dynamic fitness landscapes. In: Proceedings of the Eleventh Conference on Congress on Evolutionary Computation (CEC 2009), pp. 1613–1620. IEEE Press, Piscataway (2009)
[23] Richter, H., Yang, S.: Learning behavior in abstract memory schemes for dynamic optimization problems. Soft Comput. 13(12), 1163–1173 (2009)
[24] Richter, H.: Change detection in dynamic fitness landscapes: An immunological approach. In: World Congress on Nature Biologically Inspired Computing, pp. 719–724 (2009)

[25] Richter, H.: Memory Design for Constrained Dynamic Optimization Problems. In: Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcazar, A.I., Goh, C.-K., Merelo, J.J., Neri, F., Preuß, M., Togelius, J., Yannakakis, G.N. (eds.) EvoApplicatons 2010. LNCS, vol. 6024, pp. 552–561. Springer, Heidelberg (2010)
[26] Schulze, R., Dietel, F., Jandkel, J., Richter, H.: Using an artificial immune system for classifying aerodynamic instabilities of centrifugal compressors. In: World Congress on Nature Biologically Inspired Computing, pp. 31–36 (2010)
[27] Richter, H., Dietel, F.: Change detection in dynamic fitness landscapes with time-dependent constraints. In: Second World Congress on Nature Biologically Inspired Computing, pp. 580–585 (2010)
[28] Richter, H., Dietel, F.: Solving Dynamic Constrained Optimization Problems with Asynchronous Change Pattern. In: Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcázar, A.I., Merelo, J.J., Neri, F., Preuss, M., Richter, H., Togelius, J., Yannakakis, G.N. (eds.) EvoApplications 2011, Part I. LNCS, vol. 6624, pp. 334–343. Springer, Heidelberg (2011)
[29] Schiex, T., Verfaillie, G.: Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. International Journal of Artificial Intelligence Tools 3, 48–55 (1993)
[30] Schwarz, B., Bhandoola, A.: Trafficking from the bone marrow to the thymus: a prerequisite for thymopoiesis. N. Immunol. Rev., 209–247 (2006)
[31] Yang, S., Richter, H.: Hyper-learning for population-based incremental learning in dynamic environments. In: Proceedings of the Eleventh Conference on Congress on Evolutionary Computation (CEC 2009), pp. 682–689. IEEE Press, Piscataway (2009)