Dynamic Constrained Optimization with Offspring Repair based Gravitational Search Algorithm

Kunal Pal[1], Chiranjib Saha[2], Swagatam Das[3], and Carlos A. Coello Coello[4]
[1,2] Dept. of Electronics and Telecommunication Engg., Jadavpur University, Kolkata, INDIA
[3] Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata, INDIA
[4] Department of Computer Science (Evolutionary Computation Group), CINVESTAV-IPN, MEXICO
Emails: mail.kunalpal@gmail.com, saha.chiranjib@ymail.com, swagatamdas19@yahoo.co.in, ccoello@cs.cinvestav.mx.

***Abstract*—Dynamic Constrained Optimization Problems (DCOP) are a unique class of optimization problems where the objective function as well as the constraint functions change with respect to time. Conventional DCO algorithms involve Genetic Algorithms (GAs) accompanied by a separate constraint-handling technique e.g., a repair method, or a penalty function. However, ordinary repair methods with elitism significantly decrease the diversity of the population during the exploitation stage and penalty functions cannot properly deal with disconnected feasible regions. In this paper, we propose a new approach based on the Gravitational Search Algorithm as well as a modified version of a repair method that produces improved results. The proposed approach incorporates knowledge-reusing and knowledge-restarting in order to produce a quick recovery and faster convergence.**

I. Introduction

In constrained optimization problems [1], the basic goal is minimize the objective function subject to one or several constraint functions. These constraints increase the challenge of the problem by reducing the feasible search space. Evolutionary Algorithms (EAs) are efficient to perform unconstrained search. But in constrained environments, the search process becomes complicated due to the restriction in feasible search space. Therefore, to solve constrained optimization problems these EAs must be accompanied by some constraint handling (CH) methods.

Dynamic Optimization problems (DOPs) are a large subset of the real world optimization problems where the objective function is dynamic in nature i.e., the objective function varies with time. These problems are different from dynamic problems (also known as dynamic environments) only when "the underlying fitness landscape changes during the operation of the EA" [2]. These problems are to be solved online by an optimization algorithm as time goes on. DCOP, such as those proposed in [3] are a bit tougher, since the objective function as well as the constraint functions can be dynamic in this case. Therefore, the constraint environment itself changes causing modifications in the structure and the percentage of the feasible region. To solve these problems, the optimization algorithm must be able to track changes in the environment and adaptively modify the search strategy when a change is detected.

Recent works on DCOP has involved the triggered Hyper-Mutation Genetic Algorithm. In hyper-mutation, introduced in [4], the mutation rates are triggered to a very large value whenever a change in detected. In this way, this algorithm copes with the dynamics of the environment by introducing diversity in the population and then the algorithm adapts to new environments. However, this algorithm cannot detect a change when new global optima are exposed without changing the objective value of the previous optima. Also, determining the correct mutation rate is very difficult since it is a function of the degree of change in the environment.

Swarm based optimization algorithms has also been applied [5], [6] to solve DCOPs. Hu and Eberhart [7] applied Particle Swarm Optimization where part of the swarm is diversified by randomized relocation of the particles after a change in the environment is detected. Following a more sophisticated approach [8], the swarm can be divided into a hierarchy of sub-swarms in order to introduce diversity. But, determining the number of sub-swarms is a tough challenge and this algorithm fails to converge to an optimum when there is a large change in the objective landscape, as well.

This paper introduces an algorithm to solve dynamic constrained optimization problems efficiently using the GSA with an offspring repair technique. The proposed algorithm tracks changes in the environment separately and relocates the population all over the modified feasible region whenever environment is restructured. Thus, it can search the whole feasible space and explore new global optima irrespective of the presence of small or large changes in the fitness space. The worst particle in every generation is randomly replaced by a new particle from the feasible space which introduces diversity in the algorithm. Also, the gravitational constant is replaced by a high value whenever the population is relocated and henceforth, maintains diversity for the first few generations.

There are several CH methods available (e.g., penalty function, stochastic ranking, etc.). However, according to the No Free Lunch Theorem, no single CH method is efficient for solving all the available types of constrained optimization problems. Discarding any offspring on the basis of its constraint violation is not an efficient method, since the new particle may contain valuable information which might get lost with that particle. The penalty function, which is the most

popular CH method in current use, also fails when there are several disconnected feasible regions in the constraint space. The CH technique used here is the repair method. In this technique, if any offspring enters the infeasible region, it is replaced by a random particle between it and its closest feasible solution in the population. Therefore, the information in the infeasible particle is partially preserved.

A short introduction to the Dynamic Constrained Optimization Problem is provided in Section II. This section also discusses some real-life applications of DCOPs. Section III introduces GSA [9]. The shortcomings of conventional CH techniques and the benefits of using a repair method are discussed in Section IV. Section V describes the detection of change in an environment and Section VI shows a comparative analysis of several DCO algorithms. This section also gives a thorough discussion explaining the better performance of our algorithm. Finally, Section VII provides our conclusions and suggests some possible paths for future research.

II. Dynamic Constrained Optimization

DCOPs are a combination of a single objective function and one or several constraint functions such that at least one of them is dynamic in nature. Without loss of generality, a Dynamic Constrained Optimization Problem can be stated as follows:

$$\min_{x \in D_t \subseteq [L,U]} f(x,t) \tag{1}$$

$$G_i(x,t) \leq 0, \forall i \in \{1,2,....m\}, \tag{2}$$

where $t \in N^+$ is called time (environment) variance, $[L,U] = \{x = (x_1, x_2, ...x_n) \mid L_i \leq x_i \leq U_i, i = 1 \sim n\}$ is called search space and x=$\{x \mid x \in [L,U], G_i(x,t) \leq 0, i = 1 \sim m\}$ is called feasible space.

For $\forall x \in D_t$, if there exists a point $x^* \in D_t$ such that $f(x^*,t) \leq f(x,t)$ (or $f(x^*,t) \geq f(x,t)$ for maxima) then x^* is called the optimal point and $f(x^*,t)$ is called the optimal value for the environment t.

DCOP can be classified into two categories, combinatorial and continuous. There are miscellaneous real life applications for continuous DCOP from which optimal control of Hybrid systems [10], [11] and Dynamic systems [12] constitute more than 70%. Other than that, source identification, parameter estimation, pattern recognition and classification are other popular known fields of application for DCOP [1]. Based on the dynamic characteristics of the objective function and the constraint functions, DCOP can be of three types. The first type, where the objective function is dynamic while the constraints are static, has applications in evolvable hardware design [13], and dynamic optimization of fed-batch fermentation processes [14], among others. Next is the combination of static objective and dynamic constraint function like in hydrothermal scheduling problems [15], cargo movement problems [16], etc. The third type is one, where both of them are dynamic, and includes applications such as aerodynamic and structural design problems [17] and optimal control problems [18]–[20], among others.

III. Gravitational Search Algorithm

Let's consider a system of particles in free space; gravity minimizes the total gravitational potential energy of the system. Thus, the gravitational force field by itself, works as an optimizer. This idea is adopted in GSA. Each particle in GSA has two characteristics, viz. position and mass, where the former refers to a solution of the given problem and the latter is determined using the fitness value at that position. All the particles are attracted towards the heavier ones and the swarm ultimately converges to the optima.

From Newton's theory of gravity, the gravitational force acting between two particles is proportional to their individual mass M_i and inversely proportional to the square of the distance R between them. Hence, the force is defined as [9]:

$$F = G\frac{M_1 M_2}{R^2}, \tag{3}$$

where G is the constant of proportionality, designated as Gravitational constant. Therefore, a fair way to develop the Gravitational Search Algorithm is to randomly distribute particles over the entire search space and examine their gravitational interaction. The gravitational attraction force acting on each particle creates an acceleration a which is, given by the laws of motion:

$$a = \frac{F}{m}. \tag{4}$$

This acceleration changes the position and velocity of the particles. Consequently, these particles tend to come closer to each other eventually. In GSA, the parameters are defined in such a way that the particle with the best fitness value among the population attracts the whole swarm towards itself. In that course, some of the moving particles come closer to the optima and snatch the crown of the best particle.

Now, as the particles are virtual they don't have any real mass. They can be ranked only with respect to their fitness. Therefore, the mass assigned to the particles must be a function of their fitness value. As referred in [9], the mass assigned to each particle is defined as:

$$m_i(\tau) = \frac{fit_i(\tau) - worst(\tau)}{best(\tau) - worst(\tau)}, \tag{5}$$

where $fit_i(\tau)$ represents the fitness value of i^{th} particle at time τ and,

$$best(\tau) = \min_{j \epsilon 1,..,N} fit_j(\tau), \tag{6}$$

$$worst(\tau) = \max_{j \epsilon 1,..,N} fit_j(\tau). \tag{7}$$

The mass of the particles can be distributed over a wide range which affects their mutual gravitational interaction. It has been observed that assigning the mass after normalizing them provides a better simulation of the force acting between the particles. The normalized mass of the particles are expressed as:

$$M_i(\tau) = \frac{m_i(\tau)}{\sum_{j=1}^{N} m_j(\tau)}. \quad (8)$$

For a system with N particles, the position of the i^{th} particle is given by $X_i = (x_i^1, , x_i^d, , x_i^n)$ for $i = 1, 2, \ldots, N$ where x_i^d is the position of the i^{th} particle at the d^{th} dimension. Therefore, the distance between two particles is the Euclidian distance defined as:

$$R_{ij}(\tau) = \| X_i(\tau), X_j(\tau) \|_2 . \quad (9)$$

The Gravitational Constant in the real world is constant w.r.t. time. However, in order to enhance the convergence and exploration of the approach, in GSA the Gravitational Constant is a function of time.

$$G(\tau) = G(\tau_0) \times \left(\frac{\tau_0}{\tau}\right)^{\beta}, \quad \beta < 1, \quad (10)$$

where t_0 is the number of maximum iterations, t is the number of the current iteration and $G(t_0)$ is the final value of G. Since the value of β is less than 1, the initial value of G is very high. This helps the exploration technique and guarantees a faster convergence. In DCO, it is very important to explore the entire search space for modified or newly arrived optima after a change in the environment. At the last stages of the search, when exploitation is preferred over exploration, for better convergence, G asymptotically tends to $G(t_0)$. The parameter β is used to manipulate the value of G. Even for the same $G(t_0)$, a lower value of β ensures a higher value of the initial G. In this way the convergence of GSA is controlled.

At a specific time t we define the force acting on mass i from mass j as follows:

$$F_{ij}^d = G(\tau)\frac{M_i(\tau) \times M_j(\tau)}{R_{ij}(\tau) + \epsilon}(x_j^d(\tau) - x_i^d(\tau)). \quad (11)$$

It has been observed that use of R_i instead of R_i^2 leads to better convergence.

To perform a good trade-off between exploration and exploitation, one strategy is to reduce the number of particles and increase the number of generations. This way, only a set of heavier particles exert force on the others. To avoid convergence towards a local optimum, exploration must be preferred at the beginning. Exploitation is introduced after some generations and gradually dominates exploration. To implement this scheme, only K_{best} particles are allowed to attract other particles where:

$$K_{best} = N + ceil\left[\frac{1-N}{g_{max}-1}\right](g-1), \quad (12)$$

where, g=number of generation, g_{max}=maximum number of generation.

Therefore, the total force exerted on a single particle is given by:

$$F_i^d(\tau) = \sum_{j \in K_{best}, j \neq i}^{N} rand_j F_{ij}^d(\tau). \quad (13)$$

IV. Handling Constraints

Conventional CH techniques such as penalty functions are inefficient for solving DCOPs since they are unable to track the optima when the feasible region is disconnected and the optimum switches between these regions. On the contrary, a repair method, such as the one adopted here, allows infeasible individuals to be transformed into feasible solutions using a reference population for that matter. Thus, the particles are attracted towards the feasible region, while preserving diversity and tracking the re-structuring of the constraint space.

Repair methods were introduced by Michalewicz and Nazhiyath [21]. The main advantage of a repair method over conventional CH techniques is their representativeness. It represents a broad class of CH techniques that do not use constraint violation compensation as penalty methods do. Instead, repair methods are based on the feedback from the search process and use such information to balance feasibility in an adaptive way. On the other hand, it has been observed that repair methods are, in a way, very robust, and therefore, they fulfil the most significant requirements for dynamic constraint handling. However, the method adopted here is problem independent and designed especially for continuous search spaces.

Although the repair method was originally proposed for GA-based algorithms, we used it with GSA. In spite of being a swarm-based algorithm, the repair method provided satisfactory results in our experiments.

In a repair method, we generate first a reference population which explicitly belongs to the feasible region. Now, if S is an infeasible individual, then a random point is generated on the straight line joining S and any member R from the reference population. If the point belongs to the feasible region, then it replaces S in the main population. The reference population is updated as well if the new solution has a better fitness value than the member from the reference population.

This conventional repair method has been modified in our proposed algorithm (see Figure IV). Here, the initial population is generated entirely in the feasible region–i.e., the size of the reference population is the same as the size of the overall population. Therefore, the main population here is a null set.

Now, if the standard deviation of the population is low, then the repair method significantly decreases the diversity of the population. Therefore, the exploration process is affected. To get rid of this phenomenon, we have used a modified version of the repair method. Instead of selecting R randomly selected individuals from the reference population, we choose R individuals such that the distance between R and S is minimum. In this way, the information contained in the infeasible particle is not lost and a feasible solution is also produced. The random selection of the generated particle also gives the mechanism a heuristic touch.

1: **function** DistBasedRepair($indivS, populationP$)
2: Calculate the distance between S and all the individuals in population P

3: Take the individual R with shortest distance from S
4: $a \leftarrow U[0,1]$
5: $X \leftarrow a \times R + (1-a) \times S$ ▷ Generate random individual X in the segment between S and R
6: **if** X is $infeasible$ **then**
7: goto step 4
8: **end if**
9: **return** X ▷ X is the repaired feasible individual
10: **end function**

There are some basic advantages of this repair method with respect to a penalty method or with respect to any other conventional CH method. It has been observed that the rate of preservation of infeasible solutions is much higher (40%-50%) for repair based algorithms with respect to penalty based algorithms (10%-15%). Although for algorithms without elitism, the rate of preservation is higher, it still gives the algorithm very slow convergence which is something to be avoided.

V. Detection of Changes in Environment

DCO differs from its static counterpart by the fact that either the objective function or the constraints (or both) keep changing with time. In the benchmark problems in this area [3], the time dependence has been modelled by setting a number of function evaluations (1000 objective function evaluations) upto which the environment is static. Henceforth, our problem can be viewed as a series of distinct static constrained optimisation problems. However, in practical situations, the environment can change with any random frequency. Considering the worst case scenario to be when the environment changes very rapidly, the optimization algorithm must be able to work very fast. To achieve that, the best alternative is to use the knowledge from the previous search and that is why this algorithm uses knowledge-reuse as well as knowledge-regeneration. Now, if there is a change in the environment and it remains undetected, then the algorithm is said to fail to cope with the dynamic nature of the problem. So, the detection of change in the environment is very important and must be very fast.

The detection of changes is realized with one or a fixed set of detectors and their present and past objective values and constraint violations are compared. If they are different, then, indeed, the environment has changed. Average best fitness or some members from the population or a point (which could be random or fixed) can be used as detectors. However, if the algorithm is assisted with elitism, then the diversity of the population decreases eventually with generations. So, the population is trapped in a fixed part of the fitness landscape and if the change is not in that part, then there is a possibility that the change will be left undetected. So, this algorithm uses an entirely isolated set of detectors and compromises some objective function evaluations to achieve a quick and sensitive change detection system.

There are two types of changes that can affect the search process. One, is a change in the objective function, and the other is a change in the constaint functions. Both types of changes can be efficiently detected by establishing a set of fixed points (i.e. pivots) and then evaluating the fitness value and the amount of constraint violation after each iteration. Changes in the constraint functions not only change the boundary of the feasible region but also add (or remove) disjoint feasible regions or can change the amount of violation of any individual, while keeping the boundary of the feasible region fixed. To detect changes in the constraint landscape, we do not require any extra evaluations and, therefore, there is no limitation in this case. Hence, an arbitrary number of pivots can be assigned for detection. Now, for the former case, each time a change is detected, some evaluations are required to set the reference values for the pivots. Added to that, there will be some more evaluations required to compare the past and present values of the detectors. But, it has been observed that the number of fitness function evaluations needed for change detection is substantially small when compared to that used for optimization purposes. However, for much complicated situations, we will require a more intelligent algorithm for such change detection. Such an algorithm should consume less evaluations while still being able to detect changes fairly quickly.

VI. Results and Discussions

Real-life applications of DCOPs are numerous, e.g. dynamic aircraft scheduling [22], dynamic vehicle routing [23], video based motion capture [24], modelling of oscillatory behavior of bacterial cultures [25], modelling and control of open plate reactors [26], etc. In real-world DCO problems, the objective function and constraint functions can be combined in three different types. The first type of combination is the case where both the objective function and the constraints are dynamic, as in scheduling/resource allocation problems [17], aerodynamic/structural design problems [27], or in many optimal control problems [18] [19] [20]. In the second type of combination, the objective function is dynamic while the constraints are static, for example, in the document stream modelling problem [28], the evolvable hardware design problem [13] or the optimal control problem of fermentation processes [14]. In the third type of combination, the objective function is static and the constraints are dynamic, as can be seen in the hydrothermal scheduling problem [15], the cargo movement problem [16] and the ship scheduling problem [29].

In these three types of combinations, the alignment of infeasible regions can affect the optimization process, since multiple infeasible regions can be completely disconnected or the global optimum can be in a narrow feasible region within the infeasible region.

The test environment used here contains all these three types of real world DCOPs and makes the optimization process very challenging by narrowing the feasible space sometimes e.g., in G24_5 and G24_7, the percentage of feasible region reduces linearly from 44.61% to 7.29%. Another challenge for the algorithms is the presence of disconnected feasible region e.g., G24_3 and G24_3b, represents the first and the third type of DCOPs respectively and contains 2-3 disconnected

feasible regions in average. The performance analysis of the DCO algorithms are subject to the test problems adopted in this paper.

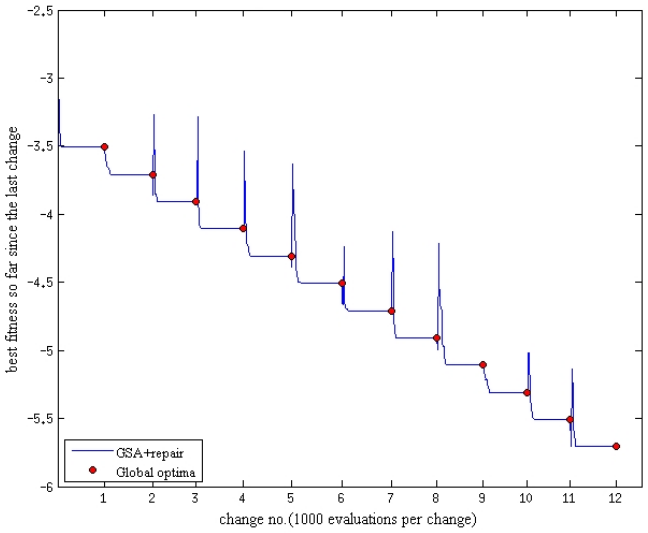

Fig. 1. $G24_3$ Fixed Objective Dynamic Constrained problem ($N = 25$)

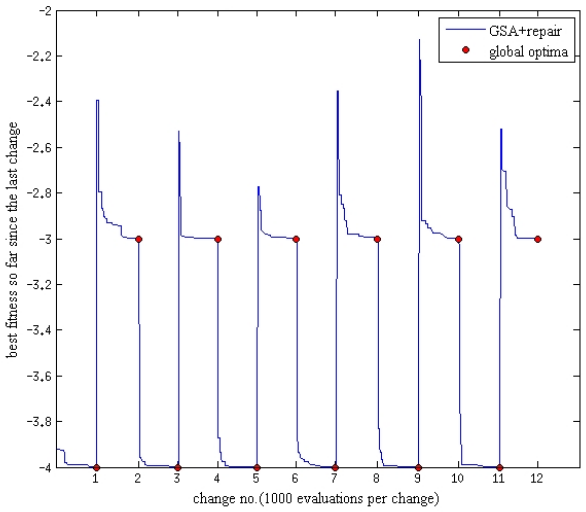

Fig. 2. $G24_{6a}$ Dynamic Objective Fixed Constraint problem (N=25)

The main goal of optimization is producing the best possible solution which is reflected in this case by the mean error. Although the problems here are dynamic in nature, a better way to calculate the error as indicated in [1] is the offline error i.e., the mean of the error in all the generations. In GSA, the K_{best} eventually decreases, which ensures a better exploitation and aims to reduce the number of function evaluations performed at each generation. A ranking based on the offline errors are given in Table I.

The dynamics of the real-life DCOPs can vary in a wide range i.e., the environment can change pretty slowly or very fast. In the test simulations, the benchmark changes the environment after every 1000 objective function evaluations but in real-life situations there is no restriction. For example, in ship scheduling problems [15] the change in the environment does not occur too often, but in training a neural network to approximate the dynamic model of unmanned aerial vehicles (UAV) [33], the change in the environment is very fast. Considering the worst case scenario, therefore, the DCO algorithm must provide quick recovery after changes in the environment. Techniques such as the triggered hyper mutation genetic algorithm enhance the mutation rate to a very high level every time a change is detected in the environment. A few extra generations are still needed to amplify the diversity to a reasonable level. In our algorithm, each time that a change is detected, a part of the old population dies out and the same number of particles are regenerated. Combined with the faster convergence of GSA, this technique provides a faster recovery.

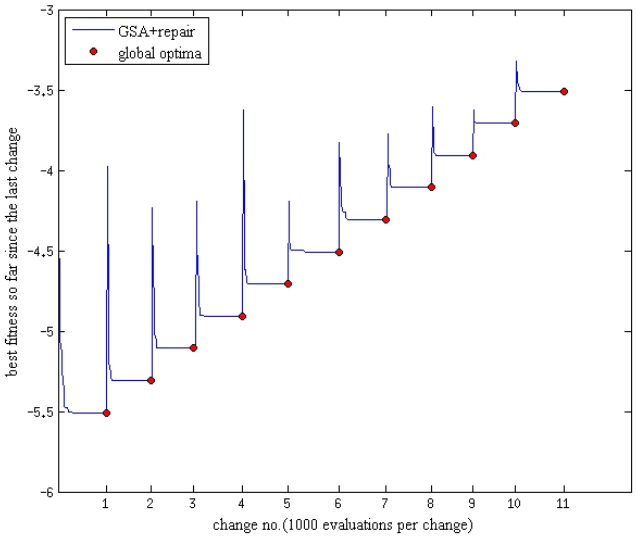

Fig. 3. $G24_7$ Fixed Objective Dynamic Constrained problem ($N = 25$)

The domain range can vary for real-life DCOPs with changes in the environment. Consequently, after a change in the environment, the explored global optima can be left out of the domain range and new global optima can be introduced when extending the domain range. Thus, the problem information available to the algorithm becomes outdated. Moreover the $HyperM$ [31] contains a parameter that controls the introduction of increased diversity after a change.

Although there can be requirements of higher or lower mutation rates depending on the severity of the change. So the tuning of $hyper\text{-}mutation$ parameter is problem specific. Under these circumstances, the $HyperM$ method may fail to produce proper results due to the shortcoming of keeping the newly introduced domain unexplored. Again, most of the standard diversity-introducting strategies do not work as effectively when integrated with CH techniques as they do in the unconstrained case.

TABLE I

OFFLINE ERRORS OF THE TESTED ALGORITHMS IN THE MEDIUM SETTINGS (POP SIZE=25; CHANGE FREQUENCY=1000 EVALUATIONS; OBJECTIVE FUNCTIONS CHANGE SEVERITY $k = 0.5$; CONSTRAINT FUNCTIONS CHANGE SEVERITY $S = 20$).

problems→	G24-u(dF, noC)			G24-1(dF, fC)			G24-f(fF, fC)		
algo.↓	mean	std	Rank	mean	(std)	Rank	mean	(std)	Rank
GA-noElit	0.298	0.051	7	0.609	0.064	8	0.676	0.085	8
RIGA-noElit	0.221	0.025	6	0.493	0.045	7	0.546	0.072	7
HyperM-noElit	0.206	0.035	5	0.361	0.065	4	0.226	0.056	6
GA-elit	0.106	0.035	2	0.459	0.057	6	0.154	0.083	4
RIGA-elit	0.149	0.025	4	0.346	0.046	3	0.178	0.051	5
HyperM-elit	0.111	0.026	3	0.384	0.065	5	0.151	0.053	3
GA+Repair	0.468	0.059	8	0.226	0.024	2	0.041	0.011	2
GSA+Repair	**0.049**	**0.004**	**1**	**0.132**	**0.015**	**1**	**0.029**	**0.012**	**1**
problems→	G24-uf(fF, noC)			G24-2(dF, fC)			G24-2u(dF,noC)		
algo.↓	mean	std	Rank	mean	(std)	Rank	mean	(std)	Rank
GA-noElit	0.464	0.064	8	0.356	0.049	8	0.159	0.041	6
RIGA-noElit	0.342	0.032	7	0.264	0.035	5	0.107	0.019	4
HyperM-noElit	0.124	0.041	5	0.257	0.045	4	0.130	0.022	5
GA-elit	0.063	0.022	2.5	0.288	0.050	7	0.073	0.017	2
RIGA-elit	0.069	0.020	4	0.246	0.037	2	0.091	0.024	3
HyperM-elit	0.063	0.012	2.5	0.253	0.043	3	**0.068**	**0.016**	**1**
GA+Repair	0.218	0.018	6	0.281	0.036	6	0.294	0.029	8
GSA+Repair	**0.047**	**0.009**	**1**	**0.182**	**0.019**	**1**	0.196	0.012	7
problems→	G24-3(fF,dC)			G24-3b(dF,dC)			G24-3f(fF, fC)		
algo.↓	mean	std	Rank	mean	(std)	Rank	mean	(std)	Rank
GA-noElit	0.760	0.099	8	0.657	0.097	8	0.886	0.179	8
RIGA-noElit	0.538	0.047	7	0.500	0.038	7	0.651	0.055	7
HyperM-noElit	0.411	0.052	6	0.459	0.069	6	0.256	0.057	6
GA-elit	0.289	0.049	4	0.457	0.084	5	0.158	0.058	5
RIGA-elit	0.308	0.048	5	0.386	0.051	3	0.167	0.048	3.5
HyperM-elit	0.283	0.050	3	0.394	0.088	4	0.158	0.051	3.5
GA+Repair	0.156	0.008	2	0.171	0.019	2	0.025	0.008	2
GSA+Repair	**0.028**	**0.004**	**1**	**0.076**	**0.009**	**1**	**0.009**	**0.007**	**1**
problems→	G24-4(dF, dC)			G24-5(dF,dC)			G24-6a(dF,2DR,hard))		
algo.↓	mean	std	Rank	mean	(std)	Rank	mean	(std)	Rank
GA-noElit	0.621	0.101	8	0.379	0.067	8	0.529	0.108	7
RIGA-noElit	0.490	0.053	7	0.293	0.046	7	0.366	0.030	3
HyperM-noElit	0.469	0.057	6	0.275	0.034	6	0.383	0.051	4
GA-elit	0.453	0.075	5	0.266	0.029	5	0.674	0.157	8
RIGA-elit	0.421	0.047	3	0.240	0.035	3	0.333	0.050	2
HyperM-elit	0.426	0.075	4	0.248	0.039	4	0.491	0.071	6
GA+Repair	0.211	0.015	2	0.236	0.024	2	0.431	0.074	5
GSA+Repair	**0.073**	**0.012**	**1**	**0.153**	**0.013**	**1**	**0.033**	**0.003**	**1**
problems→	G24-6b(dF,fC,1R)			G24-6c(dF,2DR,easy)			G24-6d(dF,2DR,hard)		
algo.↓	mean	std	Rank	mean	(std)	Rank	mean	(std)	Rank
GA-noElit	0.448	0.054	8	0.446	0.041	8	0.543	0.127	8
RIGA-noElit	0.331	0.035	3	0.329	0.039	4	0.366	0.040	4
HyperM-noElit	0.340	0.046	4	0.323	0.037	2	0.370	0.046	5
GA-elit	0.408	0.057	6	0.441	0.052	7	0.510	0.075	7
RIGA-elit	0.309	0.039	2	0.325	0.029	3	0.342	0.057	2
HyperM-elit	0.390	0.039	5	0.394	0.051	6	0.456	0.041	5
GA+Repair	0.427	0.048	7	0.390	0.038	5	0.354	0.038	3
GSA+Repair	**0.047**	**0.003**	**1**	**0.045**	**0.004**	**1**	**0.037**	**0.007**	**1**
problems→	G24-7(fF, dC)			G24-8a(dFnC,ONISB)			G24-8b(dFfC,OICB)		
algo.↓	mean	std	Rank	mean	(std)	Rank	mean	(std)	Rank
GA-noElit	0.721	0.088	8	0.426	0.050	8	0.835	0.068	8
RIGA-noElit	0.543	0.059	7	0.346	0.031	6	0.719	0.071	7
HyperM-noElit	0.495	0.053	6	0.374	0.043	7	0.681	0.072	6
GA-elit	0.316	0.053	4	0.266	0.028	2	0.662	0.056	5
RIGA-elit	0.416	0.068	5	0.304	0.028	5	0.598	0.064	3
HyperM-elit	0.315	0.062	3	0.279	0.028	3	0.608	0.071	4
GA+Repair	0.181	0.017	2	0.300	0.033	4	0.251	0.051	2
GSA+Repair	**0.018**	**0.002**	**1**	**0.202**	**0.041**	**1**	**0.192**	**0.034**	**1**

(Here dF=dynamic Function, fF=fixed Function, noC=no Constraint, fC=fixed Constraint, nDR=n Disconnected feasible Regions, OICB=Optima located at the Constraint Boundary, ONISB=Optima not in search boundary.)

But, as most of the particles are regenerated in this algorithm, after each change in the environment, it can deal with such dynamics if the knowledge of the newly shaped domain is supplied.

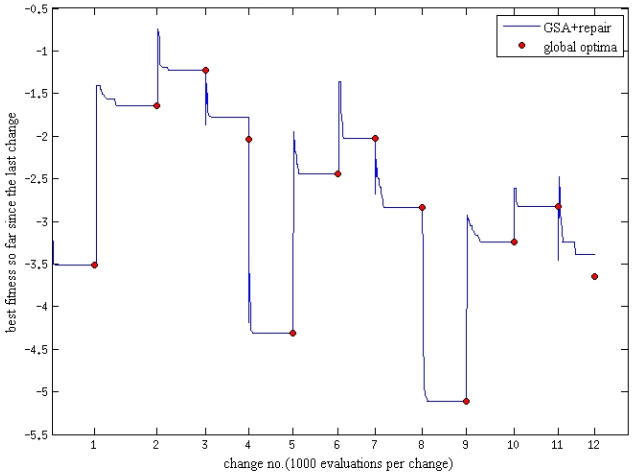

Fig. 4. $G24_4$ Dynamic Objective Dynamic Constrained problem ($N = 25$)

The change detection technique used in conventional DCOPs is based on one or on a fixed set of detectors. Hence, most of the known algorithms usually track the change based on the average best fitness [30] or on some members of the population or on a point (single detector, could be random or fixed). Although, in order to get better solution, the diversity of the population should be allowed to eventually decrease and the population can be centered in a part of the objective space. If the change occurs in some other zone, then it cannot be detected using the members of the same population. To get rid of that, this algorithm is assisted with an entirely isolated detection system. The use of a single detector is risky for obvious reasons. So, we used a detector-grid made of a fixed set of equally spaced points as reference and compared for a random member from that grid. To make the method more sensitive, the number of detectors and the comparators can be increased at the expense of function evaluations.

On the other hand, some members from the previous population remain intact at every regeneration of the population. So, if the change does not affect the global optima, then the algorithm can easily deal with that, too. The best solution in this case would be not to regenerate the population, although this algorithm cannot indulge in that. But this does not change the statistics in a significant way. Thus, this algorithm combines knowledge-reusing, that produces faster solution and helps to learn the nature of the dynamics while knowledge-restarting produces better solutions.

VII. Conclusions and Future Work

DCOPs are meant to be solved online. In this paper, we proposed an algorithm that combines GSA and a repair technique to produce better objective function values. This algorithm divides the whole problem into some discrete cases and solves them separately while they are linked together with a change detection technique to cope up with the dynamic nature of the problem.

GSA, being a swarm based algorithm, provides better control over exploration and exploitation techniques and works very well with a repair-based CH method. Therefore, this algorithm guarantees faster convergence, which is an important performance metric in DCOP. In real world applications, the change frequency can vary abruptly, and under these circumstances, it is an advantage to be prepared for the worst case scenario.

The modified repair method we proposed preserves diversity. It does not get trapped by disconnected feasible regions like the penalty function and performs better in handling constraints.

The change detection technique used in this algorithm is basic but very effective. Although it can be made more efficient by selecting more pivot points simultaneously, while compromising some objective function evaluations. Nevertheless this sort of scheme would have some problems when the change-frequency is very high. In the proposed algorithm, we have made a trade-off between these two issues. However, in our future work, we aim to design a different approach that consumes less function evaluations to detect a change in the environment.

Acknowledgements

The last author acknowledges support from CONACyT project no. 103570.

References

[1] T. Nguyen, “Classifying and characterising dynamic optimisation problems - a literature review”, tech. rep., School of Computer Science, The University of Birmingham, UK, 2007.

[2] R. W. Morrison (2004). *Designing Evolutionary Algorithms for Dynamic Environments*, number ISBN 3-540-21231-0, Springer-Verlag, Berlin.

[3] T. T. Nguyen (2008a). A proposed real-valued dynamic constrained benchmark set, Technical report, School of Computer Science, Univesity of Birmingham.

[4] H. G. Cobb (1990). An Investigation into the Use of Hypermutation as an Adaptive Operator in *Genetic Algorithms Having Continuouis, Time-Dependent Nonstationary Environments, Technical Report* AIC-90-001, Naval Research Laboratory, Washington, USA

[5] T. Blackwell (2007). Particle Swarm Optimization in Dynamic Environment, in S. Yang, Y. S. Ong & Y. Jin, eds, *Evolutionary Computation in Dynamic and Uncertain Environments*, Studies in Computational Intelligence, Springer-Verlag, NJ, USA, pp. 28-49.

[6] C. A. Liu (2008a). New Dynamic Constrained Optimization PSO Algorithm, in ICNC 08: *Proceedings of the 2008 Fourth International Conference on Natural Computation*, IEEE Computer Society, pp. 650653.

[7] X. Hu & R. Eberhart (2002). Adaptive particle swarm optimisation: detection and response to dynamic systems, in *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC2002, pp. 1666-1670.

[8] S. Janson & M. Middendorf (2006). A hierarchical particle swarm optimizer for noisy and dynamic environments., *Genetic Programming and Evolvable Machines* 7(4), 329-354.

[9] E. Rashedi, H. Nezamabadi-pour, S. Saryazdi. GSA- A Gravitational Search Algorithm, in *Elsevier, Information Sciences* 179 (2009) 2232-2248.

[10] C. de Prada ,D. Sarabia ,S. Cristea & R. Mazaeda (2008). Plant-wide Control of a Hybrid Process, International Journal of Adaptive Control and Signal Processing 22(2), 124-141.

[11] M. Fiacchini ,T. Alamo ,I. Alvarado & E. F. Camacho (2008). Safety Verification and Adaptive Model Predictive Control of the Hybrid Dynamics of a Fuel Cell System, *International Journal of Adaptive Control and Signal Processing* 22(3), 142-160.

[12] D. Dini , M. van Lent , P. Carpenter & K. Iyer (2006). Building robust planning and execution systems for virtual worlds, in *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment converence (AIIDE)*, pp. 29-35.

[13] P. Tawdross , S. K. Lakshmanan & A. Konig (2006). Intrinsic Evolution of Predictable Behavior Evolvable Hardware in Dynamic Environment, in *HIS 06: Proceedings of the Sixth International Conference on Hybrid Intelligent Systems*, IEEE Computer Society, p. 60.

[14] M. Rocha , J. Neves & A. Veloso (2005). Evolutionary Algorithms for Static and Dynamic Optimization of Fed-batch Fermentation Processes, in *B Ribeiro & others, eds, Adaptive and Natural Computing Algorithms*, Springer, pp. 288291.

[15] K. Mertens , T. Holvoet & Y. Berbers (2006). The DynCOAA algorithm for dynamic constraint optimization problems, in *AAMAS 06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, ACM, New York, NY, USA, pp. 1421-1423.

[16] P. Ioannou , A. Chassiakos, H. Jula & R. Unglaub (2002). Dynamic optimization of cargo movement by trucks in metropolitan areas with adjacent ports, Technical report, *METRANS Transportation Center, University of Southern California*, Los Angeles, CA 90089, USA.

[17] M. Andrews & A. L. Tuson (2005). Dynamic Optimisation: A Practitioner Requirements Study, in *Proceedings of the The 24th Annual Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2005)*, London, UK.

[18] M. Schlegel & W. Marquardt (2006). Adaptive switching structure detection for the solution of Dynamic Optimization Problems, *Industrial & engineering chemistry research* 45(24), 8083-8094.

[19] Y. Wang & M. Wineberg (2006). Estimation of evolvability genetic algorithm and dynamic environments, *Genetic Programming and Evolvable Machines* 7(4), 355-382.

[20] D. M. Prata , E. L. Lima & J. C. Pinto (2006). Simultaneous Data Reconciliation and Parameter Estimation in Bulk Polypropylene Polymerizations in Real Time, *Macromolecular Symposia* 243(1), 91-103.

[21] Z. Michalewicz & G. Nazhiyath (1995). Genocop III: A co-evolutionary algorithm for numerical optimization with nonlinear constraints, in D B Fogel, ed., *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, IEEE Press, Piscataway, New Jersey, pp. 647-651.

[22] J. E. Beasley , M. Krishnamoorthy , Y. M. Sharaiha & D. Abramson (2004). Displacement problem and dynamically scheduling aircraft landings, *Journal of the Operational Research Society* 55, 54-65.

[23] J. Gao & Z. Sheng (2008). Research for dynamic vehicle routing problem with time windows in real city environment, in *Proceedings of the 2008 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, vol.2, Piscataway, NJ, USA, pp. 3052 -6.

[24] M. Gleicher & N. Ferrier (2002). Evaluating Video-Based Motion Capture, in CA 02: *Proceedings of the Computer Animation*, IEEE Computer Society, Washington, DC, USA, pp. 75-80.

[25] A. J. Daugulis , P. J. McLellan & J. Li (1997). Experimental investigation and modeling of oscillatory behavior in the continuous culture of Zymomonas mobilis, *Biotechnology and Bioengineering* 56(1), 99-105.

[26] S. Haugwitz , P. Hagander & T. Norn (2007). Modeling and control of a novel heat exchange reactor, the Open Plate Reactor, *Control Engineering Practice* 15(7), 779-792.

[27] S. Padula , C. Gumbert & W. Li (2006). Aerospace applications of optimization under uncertainty, *Optimization and Engineering 7(3)*, 317-328.

[28] L. Araujo & J. J. Merelo (2007). A genetic algorithm for dynamic modelling and prediction of activity in document streams, in *GECCO 07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ACM, New York, NY, USA, pp. 1896-1903.

[29] K. Deb , U. B. Rao & S. Karthik (2007). Dynamic Multi-objective Optimization and Decision Making Using Modified NSGA-II: A Case Study on Hydrothermal Power Scheduling, in *Evolutionary Multi-Criterion Optimization, 4th International Conference, EMO 2007, Matsushima, Japan, March 5-8, 2007, Proceedings*, Vol. 4403 of *Lecture Notes in Computer Science*, Springer, pp. 803817.

[30] P. J. Angeline (1997). Tracking extrema in dynamic environments, in P. J. Angeline, R. G. Reynolds, J. R. McDonnell & R. Eberhart, eds, *Sixth International Conference on Evolutionary Programming*, Vol. 1213 of *LNCS*, Springer, pp. 335-345.

[31] S. Bird & X. Li (2007). Informative performance metrics for dynamic optimisation problems, in *GECCO 07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ACM, New York, NY, USA, pp. 18-25.

[32] J. Branke , E. Salihoglu & S. Uyar (2005). Towards an Analysis of Dynamic Environments, in H. G. Beyer & others, eds, *Genetic and Evolutionary Computation Conference*, ACM, pp. 1433 1439.

[33] A. Isaacs , V. R. Puttige , T. Ray , W. Smith & S. G. Anavatti (2008). Development of a memetic algorithm for Dynamic Multi-Objective Optimization and its applications for online neural network modeling of UAVs., in *Proceedings of the International Joint Conference on Neural Networks*, IJCNN 2008, IEEE, pp. 548-554.