ICHEA – A Constraint Guided Search for Improving Evolutionary Algorithms

Anurag Sharma and Dharmendra Sharma

Faculty of Information Sciences and Engineering
University of Canberra, ACT, Australia
{Anurag.Sharma,Dharmendra.Sharma}@canberra.edu.au

Abstract. Many science and engineering applications require finding solutions to optimization problems by satisfying a set of constraints. These problems are typically NP-complete and can be formalized as constraint satisfaction problems (CSPs). Evolutionary algorithms (EAs) are good solvers for optimization problems ubiquitous in various problem domains. EAs have also been used to solve CSPs, however traditional EAs are 'blind' to constraints as they do not exploit information from the constraints in search for solutions. In this paper, a variation of EA is proposed where information is extracted from the constraints and exploited in search. The proposed model (ICHEA for Intelligent Constraint Handling Evolutionary Algorithm) improves on efficiency and is independent of problem characteristics. This paper presents ICHEA and its results from solving continuous CSPs. The results are significantly better than results from other existing approaches and the model shows strong potential. The scope is to finding at least one solution that satisfies all the constraints rather than optimizing the solutions.

Keywords: constraint satisfaction problem (CSP), Evolutionary algorithms (EAs), Intelligent Constraint Handling Evolutionary Algorithm (ICHEA), optimization problems.

1 Introduction

Many engineering problems ranging from resource allocation and scheduling to fault diagnosis and design involve constraint satisfaction as an essential component that require finding solutions to satisfy a set of constraints over real numbers or discrete representation of constraints[3, 4, 19].Even though CSP solving is an essential computer science approaches only little research has been reported on the development of efficient and effective constraint-handling techniques (especially with respect to a plethora of new methods available for unconstrained optimization [11]).

The classical algorithms that solve CSPs include branch and bound, backtrack algorithm, iterative forward search algorithm, local search but heuristic methods such as evolutionary algorithms(EAs) have mixed successand for many difficult problems these are the only available choices [1, 4, 16]. EAs however suffer from some of its inherent problems to solve CSPs as they do not make use of information from

T. Huang et al. (Eds.): ICONIP 2012, Part I, LNCS 7663, pp. 269–279, 2012.

constraints and only blindly search in the solution space using different heuristic search algorithms [4]. Generally objective functions are designed on problem dependent penalty functions but some use generalized error measurements like *euclidean* distance from feasible regions which is applicable to continuous CSPs only.

The constraint problems can be divided into two classes: constrained optimizing problems (COPs) and constraint satisfaction problems (CSPs).The difference between these classes is that in the first an optimal solution that satisfies all constraints should be found, while in the second class any solution as long as all the constraints are satisfied is acceptable[8]. (This paper only focuses on CSPs.)

Characteristically, CSPs solved by EAsuse penalty based functions. A penalty function updates the fitness of chromosomes in EA. A penalty term is used in general for reward and punishment for satisfying and/or violating the constraints [2]. However, its main shortcoming is that penalty factors which determine the severity of the punishment, must be set by the user and their values are problem dependent[14]. Some other constraint handling approaches include expensive *repair* algorithms that promote the local search to transform infeasible solutions to feasible solutions because the feasible parents not necessarily produce feasible progenies[2].In multi-objective optimization (MOO) constraints are transformed into multiple objectives. There are many established MOO algorithms like MOGA[9], VEGA [18], NSGA and NSGAII[6]. Paredis in [17] has used co-evolution strategies that utilizes *predator-prey* model to keep two populations – one population represents solutions that satisfies many constraints while other population represents those individuals whose constraint(s) is violated by lots of individuals in the first population. This strategy requires extra computational effort to find the intersection of a line with the boundary of the feasible region.

The main contribution of this paper is to provide a variation of EAs that reduce its inherent problems by exploiting constraints in improving search. The paper is organized as follows: Section 2 briefly discusses the EA techniques used in handling constraints. Section 3 describes our proposed algorithm ICHEA – a variation of EA. Section 4 elaborates more on ICHEA from algorithmic perspective. Section 5 shows experimental results of ICHEA with other EAs used in solving CSPs. Section 6 discusses the outcomes of the experiments performed and section 7 concludes the paper by summarizing the results and proposing some further extensions to the research.

2 Constraint Handling through EAs

Traditional EAs are 'blind' to constraint as they do not extract the knowledge from the constraints but search the solution through random heuristic greedy approach [3, 5]. This causes the search engine to spend extra computational effort in searching for the solution into the wider search space without only concentrating in the restricted smaller feasible search space. Constraints can reduce the search space and it can make the heuristic search more efficient by harnessing information from constraint to guide the search engine, search in feasible search space only.Some research has been reported on working with feasible search space or searching for solution in

the boundary between the feasible and infeasible region known as strategic oscillation [10]. Normally, these are problem dependent, or require complex calculation to perform crossover and mutation to produce feasible progeny from feasible parents. Moreover, finding an initial feasible solution is itself an NP-hard problem [2].

Generally, violation count is used as a fitness function for any CSP. Depending on the strengths of constraints, individual weights can be assigned to constraints in a penalty function to calculate the fitness value. To avoid problem dependent penalty functions and to utilize some information from constraints to guide the evolutionary search many heuristic algorithms do not use violation count but use a distance function to indicate how far an individual is from the feasible regions [15]. It transforms constraint functions to a fitness function to rank individual chromosomes. This fitness function tries to progressively bring the chromosomes closer to the feasible space using the following function:

$$fitness_i(X) = \begin{cases} g_i(X), & if(g_i(X) < 0) \\ 0, & if(g_i(X) \geq 0) \end{cases} \quad (1)$$

$$e = \sum_{i=1}^{m} |fitness_i(X)| \quad (2)$$

where X is an input vector, m is the total number of constraints and g_i is i^{th} constraint function in the form of $g_i(X) \geq 0$ explained in Section 4 at Eq. (6-8). The fitness function $fitness_i$ is a measurement of *euclidean* distance of vector X from the nearest point of the feasible region where constraint g_i is satisfied. The error function e is the summation of all the fitness functions. The objective is to minimize the error value e. The solution to CSP is found when $e = 0$ where at least one solution is acceptable. Fitness value based on the distance from constraint satisfaction regions given in Eq. (1) and Eq. (2) produce good results and are independent of problems but the major drawback of such fitness functions is that they are limited to continuous domains only. They cannot be used for discrete CSPs where fitness functions typically depend on violation counts and penalty functions.

ICHEA attempts to solve CSP by utilizing as much information as possible that can be derived from constraints to guide the evolutionary search without using penalty functions and making it problem independent. It is a proposed variation of EA that does not disregard the information from constraints to produce more efficient results. This algorithm also does not require initial feasible solutions and it is also not restricted to produce feasible progenies from feasible parents.

3 Intelligent Constraint Handling Evolutionary Algorithm

Constraint satisfaction problem (CSP) is defined by an inputvector $X = \{x_1, x_2, \ldots x_n\}$in a finite search space S where each variable x_i has afinitedomain D_i. A set of constraints $\{c_1, c_2, \ldots c_m\}$ are defined in the form of functions:

$$c_i(x_1, x_2, \ldots x_n) = \begin{cases} 1, & if\ satisfied \\ 0, & if\ violated \end{cases} \quad (3)$$

Constraint satisfaction sets (feasible regions for each constraint) $\{S_1, S_2, .. S_m\}$ can also be defined where:

$$S_i = \{X \in S \mid c_i(X) = 1, 1 \leq i \leq m, i \in Z(integer\ set)\} \quad (4)$$

The solution of a CSP is $s \in S$when all the constraints $c_i$are satisfied, which can be given as:

$$\sum_{i=1}^{m} c_i(s) = m \quad (5)$$

For continuous CSPs numerical constraints can be given in two forms – equality and inequality[6, 13, 22].

$$g_i(X) \geq 0 \qquad i = 1, \ldots, k \quad (6)$$

$$h_j(X) = 0 \qquad j = k+1, \ldots, m \quad (7)$$

The equality constraints cannot be solved directly using EAs so it is converted into inequality constraint by introducing a positive tolerance value δ.

$$g_j(X) = \delta - \left|h_j(X)\right| \geq 0 \quad (8)$$

To utilize constraint satisfaction sets in an EA, a **new crossover** operator is definedin ICHEA that usesinformation from constraints rather than blindly search for the solution. The idea of *intermarriage*is incorporated intoour proposed crossover operator where both parents belong to different subpopulation sets i.e. satisfaction sets in the context of CSPs. In other words the parent vectors from different sets $S_i$and S_j are taken for crossover where $i \neq j$. It is also possible that a parent does not belong to any of the constraint satisfaction set i.e. $S - (S_1 \cup S_2 \cup \ldots \cup S_m)$. The generated offspring contains genes from both parents. The purpose is to make a "generic" offspring that tries to satisfy more than one constraint because its parents are from two different constraint satisfaction sets. The algorithm favors those offspring which satisfy more constraints by utilizing Deb's ranking scheme based on feasibility [6] to rank the population. Constraint handling techniques can fall into two categories – discrete and continuous constraint functions. Only continuous constraint functions form the scope of this paper.

3.1 Intermarriage Crossover for Continuous CSPs

In intermarriage crossover, two parents generate two offspring. This is a dual process where both parent move closer to each other one at a time and their new positions are considered as two new offspring. An offspring from two parentsthrough intermarriage is defined in a search space as a constant multiple of difference of two parent vectors as shown in Eq. (9). Initially offspring O_1 is placed at $(P_2 - P_1)/2$ (a binary traversal) then it iteratively moves closer to parent P_1 until it also satisfies the constraint(s) that P_1 satisfies and similarly offspringO_2 is designated.The iterative move can be defined as:

$$O_1 = \left(\frac{1}{2}\right)^i (P_2 - P_1) \quad (9)$$

Variable i gets incremented from 1 to a threshold Tin the sequence$\langle 1,2,\ldots,T\rangle$. The intermarriage crossover process is shown in the Fig. 1 where ✓ mark indicates possible placement for an offspring and × mark indicate the offspring vector is unacceptable in that particular position. So using the Eq. (9) the next i value is used until the offspring finds an acceptable place or a threshold value T is reached.The intermarriage crossover would cause two selected vectors (as parents) of different constraint satisfaction sets to come closer (as offspring) towards constraint boundary because the solution space lies in the overlapping boundary region. Favoring points for intermarriage that satisfy more constraints results in finding solution space quickly. As intermarriage crossover looks for feasible solution using binary search in each side of the parents, its worst time complexity is $2\log_2 T$ which represents total number of function calls (NFC) in a worst case of an intermarriage crossover. We empirically determined $T = 10$ for our experiments. Larger values like 20, 50, 100 slow down the overall process and smaller values like 1, 2, 5 reduces the capability of intermarriage crossover to locate overlapping regions.

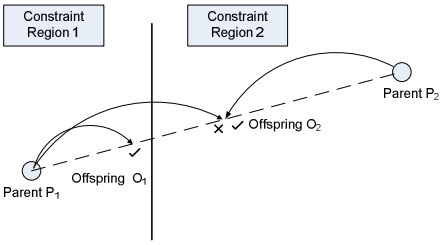

Fig. 1. Intermarriage Crossover

3.2 Locating Feasible Regions

For some problems, compared to the size of the search space the satisfaction set is relatively very small and search for a solution is difficult. So ICHEA starts with self-defined extended feasible search space of the satisfaction sets that gradually decreasesto the actual size. It helps in locating the feasible region by narrowing down the search. In the test problems search space can be extended or reduced by changing the value of δ given in Eq. (8). ICHEA starts with value of $\delta = 10^{maxN}$ that gradually decreases to $\delta = 10^{minN}$. As soon as a CSP solution with $\delta = 10^{maxN}$ is found$maxN$ is decremented by 1 and starts a new search untilδ reaches to 10^{minN}. Fig. 2 shows how constraint satisfaction set for a given constraint shrinks from $\delta = 10^{maxN}$ to $\delta = 10^{minN}$. Feasible region can be located easily for $\delta = 10^{maxN}$ in the search space then it can gradually guide the evolutionary search to the actual feasible region ($\delta = 10^{minN}$) specified by the user. $maxN$and$minN$ are usually 2 and -3 respectively. $maxN < 2$ is not very effective and $maxN \geq 2$ is able to locate the feasible regions quickly. Any value above 2 can also be chosen but it will be decremented quickly after few generations as extended feasible regions are located.

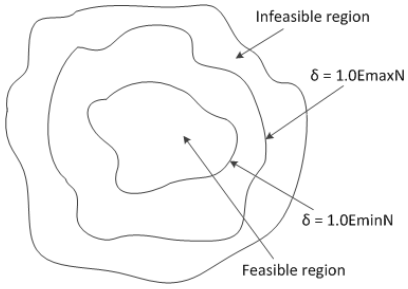

Fig. 2. Shrinking of the Feasible Region of a Given Constraint from $\delta = 10^{maxN}$ to $\delta = 10^{minN}$

4 ICHEA Algorithm

ICHEA is another variation of EA that adds constraint handling features to the standard GAs. The pseudocode can be given as:

```
chromosomes  =initializeChromosomes();
for each generation
    parents = NoveltyTournamentSelection();
    offspring = interMarriageCrossover(parents);
    Mutation(offspring);
    chromosomes = chromosomes + offspring;
    SortAndReplace();
    CheckTerminationCriteria();
  End for loop;
```

The description of subroutines is given below:

InitializeChromosomes: the population of chromosomes is randomly generated in a search space initially where 50% is boundary points specified by lower and upper bound of each dimension. The length of chromosome is same as the dimension of input variables of CSP.

NoveltyTournamentSelection: Novelty search is a new approach to heuristic search inspired by natural evolution's open-ended propensity to perpetually discover novelty. The first paper on novelty was published in 2008 by [12]. It is normally used when the objective function is deceitful for example a maze problem. The tournament selection of size 2 has been used where the winner of the two chromosomes is based on the following preferences:

1. Its novelty in the search space
2. If novelty is same then higher fitness value
3. If fitness is same then a chromosome is picked randomly

InterMarriageCrossover: This technique has been described in section 4.1.

Mutation: ICHEA uses polynomial mutation as described in [6]for real valued data.

SortAndReplace: The idea of Deb's ranking scheme based on feasibility [6] has been incorporated here. At the end of each generation, chromosomes are sorted based on the following order of preferences:

1. Number of satisfied constraints.
2. Fitness value which is the distance function from Eq. (2).

Once the chromosomes are sorted a*random death* concept is used defined in[20] to delete some predefined number of chromosomes randomly from the population. The certain top percentage of the population is spared to control the search focus.

CheckTerminationCriteria: The iteration is stopped when:

1. The maximum number of generation is reached or
2. The CSP solution is found or
3. The process is stalled by no improvement in the solution for some generations.

5 Experiments

ICHEA has been tested with canonical genetic algorithm (GA), non-dominated sorting genetic algorithm (NSGA II) [6]and hybridization of particle swarm optimization with differential evolution (PSO-DE) [13]to compare its robustness and efficiency. GA has been used as-it-is without any modification or specialization to handle constraints. NSGA II solves CSPs through multi-objective optimization and it has been shown in [6] that it has an edge over other multi objective optimization algorithms to solve COPs. PSO-DE is one of the latest proposed algorithms to solve numerical COPs very efficiently. It is able to reach to optimum solutions easily for many benchmark problems [13]. The code for GA is taken from genetic algorithms toolbox revision: 1.1.4.2, Matlab 7.0.1, NSGA II is taken from its author's website in [7] developed in C language and PSO-DE is developed in C++ language which has also been taken from its author's website in [23]. ICHEA has been developed in Java language. All algorithms have been tested on Windows XP machine with Pentium (R) i5 CPU 2.52 GHz and 3.24 GB RAM. No parallel processing or distributed environment is used for the experiments. All of these algorithms use distance function shown in Eq. (2) as a fitness function without any additional penalty functions.

Table 1. Parameter Settings

Parameters	ICHEA	GA	PSO-DE	NSGA
Population Size	25	100	100	100
Crossover rate	0.8	0.8	[0.95 1.0]	0.8
Mutation rate	0.1	0.1	-	0.1
Max generations	10,000	50,000	200,000	50, 000

Table 2. Benchmark Trigonometric Problem – H77

δ		GA	PSO-DE	NSGA II	ICHEA
10^{-1}	SR	1.00	1.00	1.00	1.00
	Best	6951 gens at 69.2s	17 gens at 0.03s	136 gens at 2.4s	8 gens at 0.3s
	Median	7428 gens at 83.3s	45gens at 0.1s	136 gens at 2.4s	22 gens at 0.64s
	Worst	9532 gens at 92.2s	221gens at 0.4s	136 gens at 2.4s	48 gens at 1.53s
10^{-3}	SR	0.00 (1/5 constraint violated)	0.20	0.00 (1/5 constraint violated)	1.00
	Best	50,000 gens at 481.0s	100,141 gens at 276.1s	50,000 gens at 914.64s	447 gens at 19.0s
	Median	-	-	-	3250 gens at 113.7s
	Worst	-	-	-	6297 gens at 211.4s

An average of 10 successive runs is taken into account to test the algorithms based on success rate (SR) and generation count to reach to the solution. SR is the rate of

successful trials for each problem i.e. $SR = successful\ trials/total\ trials$. The common efficiency measurement – the number of function calls (NFC) is not used in the experiment as ICHEA is local search intensive while other algorithms are not. Table 1 shows the parameter settings for all the algorithms. Different maximum generations have been taken as the computation speed of each algorithm varies. PSO-DE uses an additional parameter – amplification faction F that is randomly generated within [0.9 1.0].

Seven benchmark problems from CSP and COP domains[15, 21]have been resourced for the comparative test. Table 2 to Table 5 shows the CSP test results for problems H77, Chem, Broyden10, and HS109. Test problems are not elaborated due to the space limitation. The test resultsshow best, median and worst solutions for each problem in terms of SR and efficiency. Columns are left blank with "-" if either it is not applicable or no

Table 3. Benchmark Quadratic Problem – Chem

δ		GA	PSO-DE	NSGA II	ICHEA
10^{-1}	SR	0.00 (2/5 constraint violated)	0.00 (5/5 constraint violated)	0.00 (1/5 constraint violated)	1.00
	Best	50,000 gens at 515.2s	200,000 gens at 461.0s	50,000 gens at 784.1s	54 gens at 0.83s
	Median	-	-	-	238 gens at 4.66
	Worst	-	-	-	559 gens at 11.1s
10^{-3}	SR	0.00 (5/5 constraint violated)	0.00 (5/5 constraint violated)	0.00 (4/5 constraints violated)	0.30
	Best	50,000 gens at 443.2s	200,000 gens at 508.2s	50,000 gens at 825.4s	5900 gens at 196.4s
	Median	-	-	-	-
	Worst	-	-	-	-

Table 4. Benchmark Polynomial Problem – Broyden10

δ		GA	PSO-DE	NSGA II	ICHEA
10^{-1}	SR	0.00 (10/10 constraints violated)	0.40	0.00 (10/10 constraints violated)	0.80
	Best	50,000 gens at 419.0s	20,187 gens at 59.8s	20,000 gens at 524.0s	116 gens at 189.1s
	Median	-	40,205 gens at 114.1s	-	248 gens at 235.1s
	Worst	-	-	-	-

good results have been obtained. Problem HS109 is a hard problem where no good solutions are observed for the original problem. Originally this problem belongs to COP domain that has been converted to CSP by introducing error value eps which is a deflection from best known optimum solution. We used the eps value of 10% rather than usual 1% to increase the size of feasibility region.

Since constraint satisfaction is a subset of constraint optimization, all algorithms have been tested on COP benchmark problems [13, 15] as well. The objective is changed to find at least one CSP solution rather than search for an optimum solution. Table 6 shows the COP test results. In many of these problems finding a feasible solution or CSP solution is not a challenge. For example problem solutions for G01, … , G10 are obtained in less than 1 second except for problem G05. Test results have been discussed in the following section.

Table 5. Benchmark Trigonometric Problem – HS109

δ		GA	PSO-DE	NSGA II	ICHEA
10^{-1}	SR	0.00 (6/11 constraints violated)	0.00 (6/11 constraints violated)	0.00 (4/11 constraints violated)	0.70
	Best	50,000 gens at 371.9s	200,000 gens at 728.8s	50,000 gens at 1903.0s	53 gens at 71.0s
	Median	-	-	-	70 gens at 239.3s
	Worst	-	-	-	-

Table 6. COP Benchmark Test Problems Results

Prob	δ		GA	PSO-DE	NSGA II	ICHEA
G01	-	SR	1.00	1.00	1.00	1.00
		Median	2 gens at 0.03s	5 gens at 0.016s	13 gens at 0.32s	1 gen at 0.02s
G02	-	SR	1.00	1.00	1.00	1.00
		Median	2 gens at 0.01s	1 gens at 0.03s	2 gens at 0.01s	1 gen at 0.01s
G05	10^{-5}	SR	0.00 (3/5 constraints violated)	1.00	0.00 (3/5 constraints violated)	1.00
		Best	50,000 gens at 318.6s	376 gens at 0.76s	50,000 gens at 910.2	18 gens 0.40s
		Median	-	1818 gens at 3.44s	-	19 gens 0.41s
		Worst	-	1958 gens at 4.4s	-	21 gens at 0.46s

6 Discussion

The experimental results based on SR and efficiency indicate that canonical GA is generally not able to handle hard CSPs when only distance value is used as a fitness function. It must resort to some penalty functions. NSGA II also struggles in solving hard CSPs. PSO-DE generally solves a problem very quickly compared to other algorithms but only if the feasible regions are larger with positive tolerance value of 1.0E-1. For hard CSPs

with positive tolerance value of 1.0E-3 (which is generally an acceptable value) PSO-DE has low SR. Comparative results of all algorithms shows the competitiveness of ICHEA where it is able to find feasible solutions with relatively higher success rate on hard CSPs as it makes use of informationfrom constraints more effectively using a novel search operator – *intermarriage* crossover.

ICHEA does not need to have initial feasible solutions; however, it is able to maintain feasible solutions through *intermarriage*crossover in an efficient way without using any kind of repair functions. The selection of parents is based on their novelty in a search space to enforce diversity [12]. The test results show that ICHEA provides a generic EA that can solve numeric CSPs efficiently without making use of problem dependent penalty functions.

7 Conclusion

This paper has introduced a new variation of EA named ICHEA to handle CSPs. It has been shown through benchmark problems that ICHEA outperforms standard GA, NSGA II and PSO-DE in terms of higher SR and efficiency. The search mechanism of ICHEA is guided by constraints where it concentrates on the feasible regions of constraint satisfaction sets to get a solution without putting extra computational effort in searching through the whole search space.

ICHEA is a work-in-progress to create a generalized tool to solve any type of CSP. It has all the potential to be extended to work for discrete data as well because its objective function only exploits information from constraints. ICHEA is currently further developed to solve COPsas characterized and will be evaluated on benchmark problems.

References

1. Brailsford, S.: Constraint satisfaction problems: Algorithms and applications. European Journal of Operational Research 119(3), 557–581 (1999)
2. Coello Coello, C.A.: Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Computer Methods in Applied Mechanics and Engineering 191(11-12), 1245–1287 (2002)
3. Craenen, B.G.W., et al.: Comparing evolutionary algorithms on binary constraint satisfaction problems. IEEE Transactions on Evolutionary Computation 7(5), 424–444 (2003)
4. Craenen, B.G.W.: Solving constraint satisfaction problems with evolutionary algorithms. Phd Dissertation, Vrije Universiteit (2005)
5. Craenen, B.G.W., et al.: Solving constraint satisfaction problems with heuristic-based evolutionary algorithms. In: Proceedings of the 2000 Congress on Evolutionary Computation, vol. 2, pp. 1571–1577. IEEE (2000)
6. Deb, K., et al.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (2002)
7. Deb, K.: Kanpur Genetic Algorithms Laboratory, `http://www.iitk.ac.in/kangal/codes.shtml`

8. Eiben, A.E.: Evolutionary algorithms and constraint satisfaction: definitions, survey, methodology, and research directions. Presented at the (2001)
9. Fonseca, C.M., Fleming, P.J.: Genetic Algorithms for Multiobjective Optimization: Formu-lation, Discussion and Generalization. In: Proceedings of the 5th International Conference on Genetic Algorithms, pp. 416–423. Morgan Kaufmann Publishers Inc., San Francisco (1993)
10. Glover, F., Kochenberger, G.A.: Critical event tabu search for multidimensional knapsack problems. In: Osman, I., Kelly, J. (eds.) Meta Heuristics: Theory and Applications (1996)
11. Kramer, O.: A Review of Constraint-Handling Techniques for Evolution Strategies. Applied Computational Intelligence and Soft Computing, 1–11 (2010)
12. Lehman, J., Stanley, K.: Exploiting Open-Endedness to Solve Problems Through the Search for Novelty. In: Proceedings of the Eleventh International Conference on Artificial Life (ALIFE XI). MIT Press (2008)
13. Liu, H., et al.: Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. Appl. Soft Comput., 629–640 (2010)
14. Mezura-Montes, E., Coello, C.A.C.: A Survey of Constraint-Handling Techniques Based on Evolutionary Multiobjective Optimization, Departamento de Computación, Evolutionary Computation Group at CINVESTAV (2006)
15. Michalewicz, Z., Schoenauer, M.: Evolutionary algorithms for constrained parameter optimization problems. Evolutionary Computation 4(1), 1–32 (1996)
16. Müller, T.: Constraint-based Timetabling. PhD Dissertation, Charles University (2005)
17. Paredis, J.: Co-evolutionary Constraint Satisfaction. In: Davidor, Y., Männer, R., Schwefel, H.-P. (eds.) PPSN 1994. LNCS, vol. 866, pp. 46–55. Springer, Heidelberg (1994)
18. Schaffer, J.D.: Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In: Proceedings of the 1st International Conference on Genetic Algorithms, pp. 93–100. L. Erl-baum Associates Inc. (1985)
19. Shang, Y., Fromherz, M.P.J.: Experimental complexity analysis of continuous constraint satisfaction problems. Information Sciences 153, 1–36 (2003)
20. Sharma, A.: A new optimizing algorithm using reincarnation concept. In: 2010 11th IEEE International Symposium on Computational Intelligence and Informatics (CINTI), pp. 281–288. Springer, Heidelberg (2010)
21. Shcherbina, O.: The COCONUT Benchmark, http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html
22. Tessema, B., Yen, G.G.: A Self Adaptive Penalty Function Based Algorithm for Constrained Optimization. In: IEEE Congress on Evolutionary Computation, CEC 2006, pp. 246–253. IEEE (2006)
23. Wang, Y.: Yong Wang CV, http://deptauto.csu.edu.cn/staffmember/YongWang.html