On PBIL, DE and PSO for Optimization of Reinsurance Contracts

Omar Andres Carmona Cortes[1], Andrew Rau-Chaplin[2], Duane Wilson[2], and Jürgen Gaiser-Porter[3]

[1] Instituto Federal do Maranhão, São Luis, MA, Brasil
omar@ifma.edu.br
[2] Dalhousie University, Risk Analytics Lab, Halifax, NS, Canada
arc@cs.dal.ca, dwilson@gmail.com
[3] Global Analytics, Willis Group, London, UK
gaiserporterj@willis.co

Abstract. In this paper, we study from the perspective of an insurance company the *Reinsurance Contract Placement problem*. Given a reinsurance contract consisting of a fixed number of layers and a set of expected loss distributions (one per layer) as produced by a Catastrophe Model, plus a model of current costs in the global reinsurance market, identifying optimal combinations of placements (percent shares of sub-contracts) such that for a given expected return the associated risk value is minimized. Our approach explores the use bio-inpired metaheuristics with the goal of determining which evolutionary optimization approach leads to the best results for this problem, while being executable in a reasonable amount of time on realistic industrial sized problems.

Keywords: Reinsurance Analytics, Reinsurance Contract Placement, Particle Swarm Optimization, Differential Evolution, Population-Based Incremental Learning, Financial Risk, Optimization.

1 Introduction

Risk hedging strategies are at the heart of prudent risk management. Individuals often hedge risks to their property, particularly from infrequent but expensive events such as fires, floods and robberies, by entering into risk transfer contracts with insurance companies. Insurance companies collect premium from those individual with the expectation that at the end of the year they will have taken in more money than they have had to pay out in losses and overhead, and therefore remain profitable or at least solvent. Perhaps not surprisingly insurance companies themselves try to hedge their risks, particularly from the potentially enormous losses often associated with natural catastrophes such as earthquakes, hurricanes and floods. Much of this hedging is facilitated by the global "property cat" reinsurance market [1], where reinsurance companies insure primary insurance companies against the massive claims that can occur due to natural catastrophes.

Analytics in the reinsurance market is becoming increasingly complex for at least three reasons. Firstly, factors like climate change are skewing the data in ways that are not fully understood making experience a less useful guide in decision making. Secondly, the global distribution of economic activity is changing rapidly with key supply-chain now having significant exposure to parts of the world where catastrophic risk is less well understood. For example, few in 2011 understood that a Thailand flood event could cost $47 Billion USD in property losses and cause a global shortage of hard disk drives that lasted throughout 2012. Lastly, there is a tendency for risk transfer contracts to become ever more complex, in large part by increasing the number of sub-contracts (called layers) that make up a contract. This in turn makes it increasingly important to have good computational tools that can help underwriters understand the interaction between layers and to decide on placement percentages, that is which layers to buy and how large a share or percentage of them to buy, in order to minimize risk for a given expected return.

In this paper, we study from the perspective of an insurance company the *Reinsurance Contract Placement problem*. Given a reinsurance contract consisting of a fixed number of layers and a set of expected loss distributions (one per layer) as produced by a Catastrophe Model [2], plus a model of current costs in the global reinsurance market, identifying optimal combinations of placements (percent shares of sub contracts) such that for a given expected return the associated risk value is minimized. Our approach is to explore the use of metaheuristics (evolutionary and swarm algorithms) with the goal of determining which approach leads to the best results for this problem, while being executable in a reasonable amount of time of realistic industrial sized problems.

There are many bio-inspired metaheuristics that can be applied to optimize problems like this, such as Particle Swarm Optimization (PSO) [3], Differential Evolution (DE) [4, 5], Genetic Algorithms (GA) [6], Evolution Strategies (ES) [7] and Population-Based Incremental Learning (PBIL) [8]. Indeed, the broader area of computational finance is a field that has been gaining attention lately in the evolutionary computation community driven by the increasing availability of financial data for analysis and improvements in computer processing power [9]. Some notable examples of metaheuristics in computational finance include [9], [10], [11], [12].

Recently, risk and reinsurance problems have also been tackled using bio-inspired algorithms such as in [13], [17] and [14]. Here the focus has been on stop loss and ruin predictions, a somewhat different problem than the contract placement problem studied in this paper. The initial work on contract placement [15] which has been applied in an industrial setting used a parallel discretized enumeration method. Unfortunately, while this method worked well when the number of layers was small (for example 2-5 layers), it experienced exponentially growing runtimes as the number of layers is increased. For instance, a problem with just 7 layers and using a discretization of 5% requires more than a week to be solved using an R-based implementation of this method, while problems of more than 7 layers or finer discretization might run for months or

years and are therefore practically infeasible. Initial work addressing the Reinsurance Contract Placement problem using evolutionary techniques was described in [16]. The approach taken was to compare the Population-Based Incremental Learning (PBIL) [8] method to the previously studied enumeration method to try and determine if the evolutionary method could find results that were comparable in quality to the exact enumeration approach, and if the use of PBIL would allow larger problems, that is those with more layers, to be solved in a feasible amount of time. While [16] demonstrated that PBIL worked for this problem it generated as many questions as it answered. For example, 1) is PBIL the best approach or would newer evolutionary methods like PSO, or DE be better? 2) How good are the results in high dimensions given that we have no other method to compare against?, and 3) what values for key parameters like number of iterations or population size work best for each method and at what point do the benefits of larger values (and corresponding larger run-times) diminish? It is these questions that this paper sets out to answer. In the remainder of this paper, we first formally define our reinsurance contract placement problem in Section 2. Then we describe the evolutionary methods PSO, DE and PBIL in Section 3. Thereafter, we present a detailed performance analysis comparing our results in terms of quality and performance on real-world data, in Section 4.

2 The Reinsurance Contract Placement Problem

Insurance organizations, with the help of the global reinsurance market, look to hedge their risk against potentially large claims, or losses [1]. This transfer of risk is done in a manner similar to how a consumer cedes part of the risk associated with their private holdings. However, unlike the case of the consumer, who is usually given options as to the type of insurance structures to choose from, the insurer has the ability to set its own structures and offers them to the reinsurance market. Involved in this process are decisions around what the type and the magnitude of financial structures, such as deductibles and limits, as well as the amount of risk the insurer wishes to maintain. The deductible describes the amount of loss that the insurer must incur before being able to claim a loss to the reinsurance contract, the limit describes the maximum amount in excess of the deductible that is claimable and the placement describes the percentage of the claimed loss that will be covered by the reinsurer.

In the reinsurance placement problem an insurer given a fixed number of layers and loss distributions is then faced with the problem of selecting an optimal combination of placements. As with most financial structures, the central problem is in selecting an optimal proportion, or placement, of each layer such for a given expected return on the contracts the associated risk is minimized. This means, from the perspective of the insurer, they wish to maximize the amount claimable for a given risk value. In doing so they minimize amount of loss the insurer may face in a year. This formulation leads to a optimization problem as depicted in Equation 1.

$$\begin{array}{ll} maximize & VaR_{\alpha}(\mathbf{R}(\pi)) \\ s.t. & E(\mathbf{R}(\pi)) = a \end{array} \quad (1)$$

Given that the expected return a is specified in Equation 1 we can rewrite it as a Pareto Frontier problem as shown in Equation 2, where q is a risk tolerance factor greater than zero. More details about the math involved in this particular optimization problem can be seen in [1], [20] and [16].

$$maximize \;\; VaR_{\alpha}(\mathbf{R}(\pi)) - qE(\mathbf{R}(\pi)) \quad (2)$$

3 Evolutionary/Swarm Algorithms

Evolutionary/Swarm algorithms are population-based stochastic algorithms that originate from nature and provide attractive features for solving both continuous and discrete problems [24]. In this section we briefly describe the three meta-heuristics we will be evaluating for treaty placement problem.

3.1 Differential Evolution

The Differential Evolution (DE) was proposed by Rainer Storn and Kenneth Price in 1995 [4, 5] to solve optimization problems [21]. The basic structure of the approach is given in the Algorithm 1, in which F is the scaling vector within the domain $[0, 2]$ and CR is the crossover rate. Initially, a population of real-coded individuals $X_i^D = (x_i^1, x_i^2, ..., x_i^D)$ is randomly created within the domain $[a_i^D, b_i^D]$ where D represents the problem dimension. Then a vector of differences is created based on the equation $x_i^{'} = x_i^3 + F \times (x_i^2 - x_i^1)$, where three member of the population are selected at random, x_i^1,x_i^2 and x_i^3. As we can see, F is used to weight the contribution of the vectors x_i^2 and x_i^3. This calculation is commonly referred as mutation.

Actually, each gene of an individual(n) is chosen taking into account the Crossover Rate (CR), *i.e.*, if the random number is less than CR then the new gen assumes the value computed by the vector of differences, otherwise the new gene is the same of x_i, where i is the index of the individual that can be replaced in the current population. The new individual will replace the current one only if the new one has the best fitness. This strategy is called DE/rand/1/bin. If the best individual is used for creating the vector of differences the strategy is called DE/best/1/bin.

3.2 Particle Swarm Optimization

The particle swarm optimization was firstly proposed by Kennedy and Eberhart [3] also in 1995. The algorithm consists of particles that are placed into the search space. Each particle moves combining some aspects of its own history position and the global position. All particles move around the search space and probably the swarm will move towards the potential optimum in the next iterations.

```
Generate a population X of size n within the domain [a_i, b_i]
for i = 1 to pop_size do
    Choose 3 individuals of population x_i^1, x_i^2 and x_i^3
    x' = x^3 + F × (x^2 − x^1)
    for j = 1 to D do
        Chose a number r at random within [0, 1]
        if (r < CR) then
            n_ij = x'_j
        else
            n_ij = x_j
        end
    end
    if (f(n_i) < f(x_i)) then
        x_i = n_i
    end
end
```

Algorithm 1: Differential Evolution(DE)

A particle represents a position in the search space as $X_i^D = (x_i^1, x_i^2, ..., x_i^D)$. Further, a particle has a velocity $V_i^D = (v_i^1, v_i^2, ..., v_i^D)$ which is used to determine its new position in the next iteration, where D represents the problem dimension. The new position is determined by means of the Equations 3 and 4, where w represents the inertia weight, c_l and c_2 are acceleration constants, r_l and r_2 are random number in the range $[0, 1]$, p_i^d is the best position reached by the particle P, and g^d is a vector stores the global optima of the swarm so far.

$$v_i^d = w \times v_i^d + c_1 r_1 \times (p_i^d - x_i^d) + c_2 r_2 \times (g^d - x_i^d) \tag{3}$$

$$x_i^d = x_i^d + v_i^d \tag{4}$$

The Algorithm 2 outlines how PSO works. Initially, the swarm is created at random, where each particle has to be within the domain $[a_i^d, b_i^d]$. Then particles are evaluated in order to initialize the P matrix and the g^d vector, which are the best experience of each particle and the best solution that has been found up to now, respectively. Thereafter, the velocity and the position of a particle are updated within a loop that obeys some stop criteria.

3.3 Population-Based Incremental Learning

Population based incremental learning (PBIL) was first proposed by Baluja [8] in 1994. In the original version of the algorithm, the population were encoded using binary vectors and an associated probability vector, which was then updated based on the best members of a population. Unlike other evolutionary algorithms, a new population is generated at random using the updated probability vector for each generation. Since Baluja's initial work, extensions to the al-

```
Generate a swarm of particles X of size s from [a_i^d, b_i^d] ;
for i = 1 to swarm_size do
    Evaluate swarm;
    Update the best position g
    Update p of the particles
    for j = 1 to D do
        Update velocity V using Equation 3
        Update position X using Equation 4
    end
end
Verify if the current g is better than the best of the current swarm
```

Algorithm 2: Particle Swarm Optimization (PSO)

gorithm have been proposed for continuous and base-n represented search spaces [19, 22].

Here we substitute the intervals for equidistant increments in the lower and upper bounds of the search space. The Algorithm 3 describes the discretized PBIL (DiPBIL) method used in this paper in terms of the following tunable parameters: I = Number of Increments (*i.e.* the discretization), LR_2 = Learning Rate in base 2, NLR_2 = Negative Learning Rate in base 2, M_R = Mutation Rate, M_S = Mutation Shift and q = Number of best results to be used in updating. In the same spirit as the original PBIL, the probability matrix is initialized with all increments having an uniform distribution and is updated after every generation with the best combinations member (see Algorithm 1). The updating of each vector in the matrix, however, is done using the base-n method, with an adjusted learning rate and updating function [23]. To ensure more population diversity from across generations, the probability matrix is updated with best member from previous generations as well as the top q members from the current generation. This modifies the updating process as shown in Equation 5, where LF_{ijk} is the i^{th} learning factor, as described in [23], for the k^{th} best result for the j^{th} variable.

```
for i = 1 to pop_size do
    Generate a population X of size n from P_ij ;
    Evaluate f = fun(X);
    Find x_G^best from the current and previous populations;
    Find x_i^best for top q-1 members of the current population;
    Update P_ij based on x_G^best ∪ x_i^best using LR_N and NLR_N;
end
```

Algorithm 3: DiPBIL

$$p_{ij}^{NEW} = \sum_{k=1}^{q} p_{ij}^{OLD} \frac{LF_{ijk}}{q} \tag{5}$$

4 Experimental Results

In this section we compare the reinsurance contract optimization technique against the three algorithms discussed previously, using an anonymized 7 layered real world data set composed by information such as: recoveries, reinstatements, loss table and rate on line (rol). Further, the level of discretization is 5%. Each algorithm was executed 31 trials, thus we can guarantee that the distribution of the outcomes of the experiments follows a normal distribution (central limit theorem) [25], allowing us to make parametric tests. Further, the test has been conducted considering three different number of iterations (500, 1000, 2000) and three population sizes (100, 200, 400), leading the complete experiment to a 837 executions. The parameters were chosen empirically and all tests have been done using R version 2.15.0 and RStudio on a Windows 7 64-bit Operating System running on an Intel i7 3.4 Ghz processor, with 16 GB of RAM. The PBIL algorithm was completely implemented in R language, whereas DE and PSO were obtained from R packages. It is important to notice that the DE package for R uses the strategy DE/best/1/bin. Moreover, the PSO package is based on the implementation of SPSO 2007 [26].

4.1 Quality Analysis

The quality analysis comprises two parts. The first one makes a comparison within each algorithm, *i.e.*, we have tried to identify how the changes on both the number of iterations and the population size affect the precision of a particular algorithm. The second one compares the quality of the solutions between algorithms. All evaluations are supported by Analysis of Variance (ANOVA) and Tukey test. Furthermore, the experiments aim to answer all questions which were done in Section 1.

Comparison Within Metaheuristics Table 1 presents the mean (average), the best, the worst, and the standard deviation of the risk (in dollars) for the algorithms considering a given expected return, where the best results are emphasized. The mean represents the average on 31 executions and results going toward zero mean lower risk, therefore, better results. Doing so, we are answering the second question. Each algorithm was evaluated for varying population size and number of iterations. The results given by DE were omitted because no differences were found, the outcome -1014986645 was reached regardless the increasing on both the number of iterations and the population size. Thus, considering these results we can state the following observations: (i) The DE algorithm is not sensitive neither to the number of iterations nor to the population size,

getting stuck in a local optima; (ii) PSO got some good results, however it can not evolve properly as long as we increase both the number of iterations and the population size, reaching the best value (-1014569720) at least once only with a population size of 100; (iii) PBIL evolves properly and reaches the best solution at least once in all configurations, allowing to find out a good pareto frontier if necessary.

Table 1. Results in terms of quality for PSO and PBIL

PSO			
500 iterations			
	100 pop	200 pop	400 pop
Mean	**-1014797783**	-1014914071	-1014894134
Worst	**-1014986645**	-1016020335	**-1014986645**
Best	**-1014569720**	-1014694720	-1014699862
Stdev	157117.8977	327431.8272	**136276.6568**
1000 iterations			
Mean	**-1014848734**	-1014912637	-1014931139
Worst	-1014986645	**-1014986645**	**-1014986645**
Best	**-1014569720**	-1014699862	-1014699862
Stdev	157584.0928	127561.8282	**115174.7963**
2000 iterations			
Mean	**-1014964318**	-1014977394	-1014977394
Worst	-1016020335	-1014986645	**-1014986645**
Best	**-1014694720**	-1014699862	-1014699862
Stdev	227266.6245	**51507.73476**	**51507.73476**
PBIL			
500 iterations			
	100 pop	200 pop	400 pop
Mean	-1015360605	-1015127063	**-1014956575**
Worst	-1016280747	**-1016176585**	**-1016176585**
Best	**-1014569720**	**-1014569720**	**-1014569720**
Stdev	723154.6754	552965.2035	**432175.8289**
1000 iterations			
Mean	-1015142297	-1015028435	**-1014924013**
Worst	-1016176585	-1016176585	**-1016020335**
Best	**-1014569720**	**-1014569720**	**-1014569720**
Stdev	617360.1748	503103.4937	**347988.6081**
2000 iterations			
Mean	-1015253346	-1015022019	**-1014959747**
Worst	-1019014085	-1016020335	**-1014986645**
Best	**-1014569720**	**-1014569720**	**-1014569720**
Stdev	904081.2636	364027.6174	**104119.1131**

Comparison Between Metaheuristics The purpose of this experiment is to compare the performance, in terms of quality, between algorithm, allowing us to answer the first question. In order to do so, we define the number of iterations and vary the population size. Figure 1 shows the average result of each algorithm, where the graphs depict 500, 1000 and 2000 iterations, respectively. Considering the number of iterations, the PSO algorithm presented the best overall results using 500 iterations. An interesting thing to noticed is that as long as we increase both the iteration number and the population size the algorithms tends to present more similar results, however three observations have to be made: (i) PBIL shows clearly how evolve itself as long as we change the population size; (ii) PSO starts presenting better solutions than the other algorithms in the initial configurations, nonetheless the algorithm worse in terms of quality when the population size changes; and (iii) as previously mentioned, DE gets trapped in a local optima. In this context, if we applied an ANOVA test in all of those combinations, the statistical meaning start disappearing when 2000 iterations and population size of 200 are used which ends up answering the third question. In other words, using a population size of 200 or 400 leads to similar outcomes. Moreover, extending this statistical evaluations to the other configurations we will see that PSO and DE some times provide similar quality of solution, whereas PBIL improves the results based mainly on the variation of the population size.

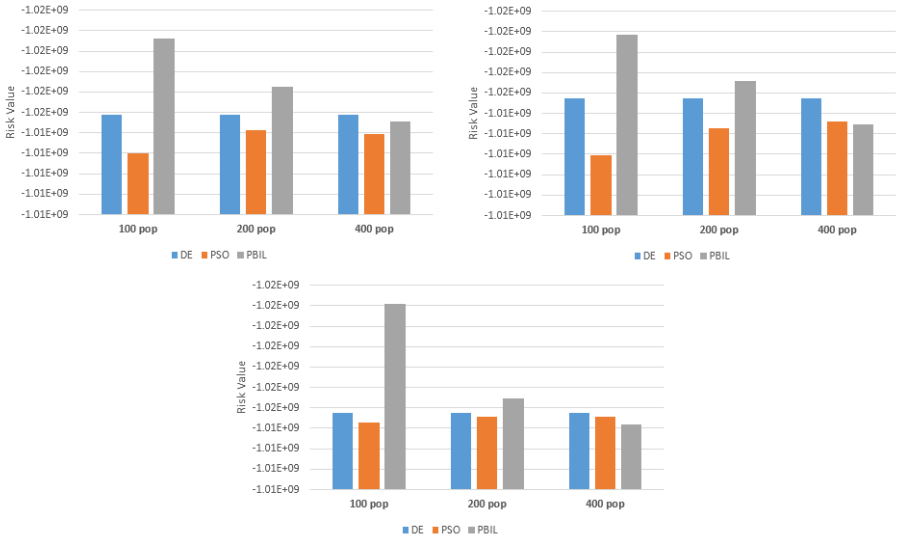

Fig. 1. Results of the different AEs for 500, 1000 and 2000 iterations, respectively

4.2 Performance

It is important to be aware that both DE and PSO have implementations in C language linked with R. Thus, we have used the compiler package [4] from R just in our PBIL code in order to improve its performance as well. Figure 3 presents the performance of the algorithms in terms of elapsed time.

It is clear that DE has the best performance in terms of time. On the other hand, it is not possible to identify if the difference between PSO and PBIL is significant. Thus, in order to compare theses algorithms we did a two-tailed t-test with $\alpha = 0.01$, where t has to be in the range $[-1.645, 1.645]$ for accept the null hypothesis (h_0) we consider as "there are no differences between means". As illustrated in Table 2, we can observe that the null hypothesis is rejected in four cases as following: (i) 500 iterations and population size equals to 100: **PBIL**; (ii) 1000 iterations and population size equals to 200: **PSO**; (iii) 1000 iterations and population size equals to 400: **PSO**; and, (iv) 2000 iterations and population size equals to 100: **PBIL**.

Looking at the results we can state the compiler package is more efficient in compiling the outer loop than the inner one, which deal with the population size, this might be the reason why PBIL is faster with small populations and higher number of iterations. Anyway, the PBIL presented a good performance because it is not written in C, but in pure R.

Table 2. A t-test between PSO and PBIL

500 iterations						
	100pop	Stdev	200pop	Stdev	400pop	Stdev
PSO	82.1648	9.7339	159.2845	7.4825	320.1658	13.57
PBIL-C	77.39	8.3282	160.74	46.8142	350.26	101.65
t	**2.0752**		-0.1709		**-1.6339**	
1000 iterations						
PSO	171.5858	12.2129	333.6309	17.7405	656.7248	21.46
PBIL-C	166.58	34.5330	368.35	102.0374	758.53	216.93
t	0.7609		**-1.8665**		**-2.6002**	
2000 iterations						
PSO	414.0819	75.7310	738.6980	77.8075	1475.4380	92.61
PBIL-C	344.11	86.4658	703.9	237.0902	1534.6525	444.91
t	**3.3894**		0.7764		-0.7254	

Inspite of using packages, parallel computing represents a viable alternative in order to speedup the pareto frontier calculation because points are computed independently (one per given expected return). In this context, Figure 2 shows the speedup obtained for each algorithm increasing the thread count. The experiment was conducted in a SunBlade server x6440, with four Quad-core AMD

[4] The compiler package improves the performance of R code creating a byte-code.

Opteron 8384 (2.7GHz) processors and 32 GB Ram, running Red Hat Enterprise Linux 4.8.

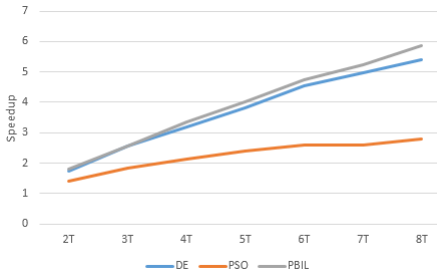

Fig. 2. The achieved speedup increasing the thread count

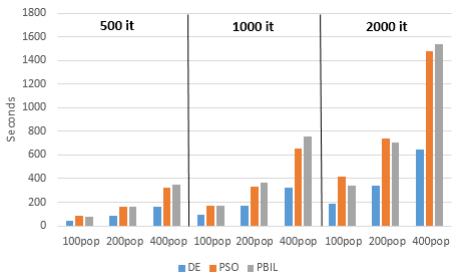

Fig. 3. Performance of the evolutionary algorithms

As we can see in the Figure 2, the best speedup was reached by the PBIL algorithm, where 8 threads leaded to a speedup close to 6. DE and PBIL had similar outcomes until 4-5 threads and then PBIL start improving a little faster. Whereas PSO presented the worse efficiency when the number of threads were increased.

5 Conclusions

This paper presented an evaluation of three different algorithms (Differential Evolution, Particle Swarm Optimization and Population-Based Incremental Learning) optimizing the problem of the Reinsurance Contract, answering relevant questions. Future implementations include a real multi-objective PBIL, PSO and DE versions, *i.e*, optimizing the risk value and the expected return at the same time. Furthermore, we plan to extend the PBIL approach evaluating the performance gains achievable with an optimized C/OpenMP implementation.

References

1. Cai, J. et al. (2008). Optimal reinsurance under VaR and CTE risk measures. *Insurance: Mathematics and Economics*, 43, 185-196.
2. Grossi, P. and Kunreuther, H. Catastrophe Modeling: A New Approach to Managing Risk, International Series on Risk, Insurance and Economic Scurity, Springer, 2005.
3. Kennedy, J.; Eberhart, R., "Particle swarm optimization," Neural Networks, 1995. Proceedings., IEEE International Conference on , vol.4, p. 1942-1948, 1995
4. Storn, R. and Price, K., Differential Evolution A simple and efficient adaptive scheme for global optimization over continuous spaces, Technical Report TR-95-012, March 1995, ftp.ICSI.Berkeley.edu/pub/techreports/1995/tr-95-012.ps.Z

5. Storn, R. and Price, K., Minimizing the real functions of the ICEC96 contest by differential evolution, Proc. of IEEE International Conference on Evolutionary Computation, Nagoya, Japan, 1996.
6. Michalewicz, Z., Genetic Algorithms + Data Structure = Evolution Programs, 3 ed, Springer, 1996.
7. Yao, X. and Liu, Y. and Lin, G., Evolutionary programming made faster, IEEE Transactions on Evolutionary Computation, v.3, n.2, pp. 82-102, 1999.
8. Baluja, S. Population based incremental learning. *Technical Report*, Carnegie Mellon University.
9. Edward Tsang, P. K. and Martinez-Jaramillo, S. Computational finance feature article. IEEE Computational Intelligence Society, 2004.
10. Gilli, M. and Schumann, E., Heuristic optimisation in nancial modelling. COMISEF wps-007, 2009.
11. Maringer, D. G. and Meyer, M., CSmooth transition autoregressive models: New approaches to the model selection problem. Studies in Nonlinear Dynamics and Econometrics, 12(1):119, 2008.
12. Krink, T. and Paterlini, S., Multiobjective optimization using Differential Evolution for real-world portfolio optimization. Computational Management Science, 8:157179, 2011.
13. Shapiro, A. F. and Gorman, R. P., Implementing adaptive nonlinear models, Insurance: Mathematics and Economics, Volume 26, Issues 23, pp. 289-307, 2000.
14. Salcedo-Sanz, S. and Carro Calvo, Leopoldo and Claramunt Bielsa, M. and Castañer, A. and Marmol, M. An Analysis of Black-Box Optimization Problems in Reinsurance: Evolutionary-Based Approache. Available at SSRN: http://ssrn.com/abstract=2260320 or http://dx.doi.org/10.2139/ssrn.2260320, 2013.
15. Mistry, S. (n.d.), et al. Parallel Computation of Reinsurance Models. Unpublished Manuscript.
16. Cortes, O. A. C. and Rau-Chaplin, A. and Wilson, D. and Gaiser-Porterz, J. "Efficient Optimization of Reinsurance Contracts using Discretized PBIL", In Proceedings of Data Analytics, London, 2013.
17. Posík, P. and Huyer, W. and Pál,A comparison of global search algorithms for continuous black box optimization, Evolutionary Computation, vol. 20, pp. 509-541, 2012.
18. Sebag, M. and Ducoulombier, A., Extending Population-Based Incremental Learning to Continuous Search Space, LNCS, v. 1498, p. 418-427, Springer, 1998.
19. Bureerat, S., Improved Population-Based Incremental Learning in Continuous Spaces, Soft Computing in Industrial Applications, p. 77-86, Springer, 2011.
20. Mitschele, A. and Oesterreicher1, I. and Schlottmann, F. and Seese1, D., Heuristic optimization of reinsurance programs and implications for reinsurance buyers, International Conference of the German Operations Research Society 2006.
21. C. Sun, H. Zhou, L. Chen, Improved differential evolution algorithms, IEEE International Conference on Computer Science and Automation Engineering , vol. 3, p.142-145, 2012.
22. Yuan, B. and Gallagher, M. Playing in continuous spaces: Some analysis and extension of population-based incremental learning, *CEC2003, CA, USA*, 443-450, 2003.
23. Servais, M.P., Jager, G. and Greene, J.R. Function optimisation using multi-base population based incremental learning. *PRASA 97, Rhodes University*, 1997.
24. Pehlivanoglu, Y.V., A New Particle Swarm Optimization Method Enhanced With a Periodic Mutation Strategy and Neural Networks, Evolutionary Computation, IEEE Transactions on , vol.17, no.3, pp.436,452, June 2013

25. Schefler, B., Statistics: Concepts and Applications, Benjamin-Cummings Pub. Co., 1988.
26. Clerc, M. A method to improve Standard PSO, Open access archive HAL, 2009, Available at http://hal.archives-ouvertes.fr/hal-00394945, Last Visit: 06-Jun-2013.