REINSURANCE ANALYTICS USING SERIAL AND PARALLEL COMPUTATION ON THE MULTIOBJECTIVE EVOLUTIONARY ALGORITHM SPEA2

Omar Andres Carmona Cortes*, Andrew Rau-Chaplin†

*Departamento de Computação
Instituto Federal do Maranhão
São Luis, MA, Brazil

†Faculty of Computer Science
Dalhousie University
Halifax, NS, Canada

Emails: omar@ifma.edu.br, arc@cs.dal.ca

Abstract— This paper presents a novel and efficient application of the SPEA2 on reinsurance contract optimization considering the perspective from an insurance company. The reinsurance operation aims to transfer the risk taken by an insurance company, usually against natural catastrophes, to a bigger corporation. The process of reinsurance is similar to that one where a client wants to insure his properties upon the payment of a premium. Then, the insurance company sells his portfolio using the reinsurance market aiming to maximize the expected return and, at the same time, to maximize the risk hedged to the reinsurance company. This problem is naturally multi-objective; consequently, the SPEA2 algorithm appears as an attractive approach to tackle the problem. Results show that the SPEA2 can obtain better outcomes than a sophisticated algorithm called enhanced MO-PBIL in terms of hypervolume. A parallel version based on the master-slave model showed that a speedup of 2.44 can be reached using eight cores and 500 iterations in a 15 layered problem, and 3.22 using eight cores and 500 iterations in a 30 layered problem, with little effort to parallelize the application.

Keywords— Reinsurance, evolutionary algorithms, SPEA2, multiobjective.

1 Introduction

The reinsurance contract optimization is a common problem in the reinsurance market, in which an insurance company wants to hedge its risk against massive claims to a more prominent company. The process is similar to that one where a client wishes to insure his property against natural disasters such as hurricanes and earthquakes. Figure 1 depicts how the flow between clients, insurance companies, and reinsurance companies happens.

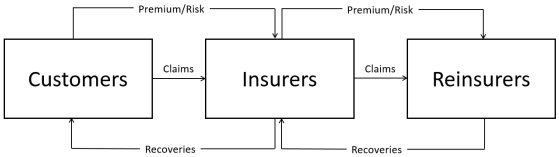
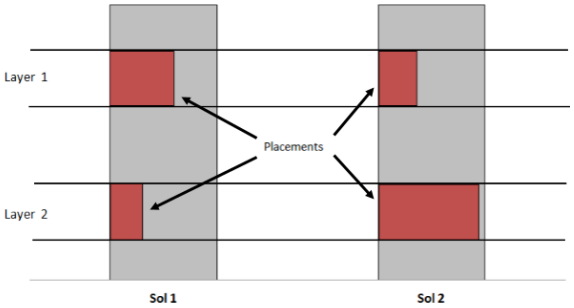


Figure 1: Reinsurance flow

A reinsurance contract is devised in term of layers, which are composed of limits, deductible, and shares (also known as placements). The limit is the maximum value that is paid in case of a claim. The deductible is the minimum financial damage that the insurance company has to afford to receive the payment. Shares are the percentage of losses that a layer will cover. Figure 2 illustrates how these concepts are represented in a contract.

Usually, a contract is devised by multiple layers transforming the optimization process into a considerably more complex problem. When ready, the contract can be negotiated in the reinsurance market aiming to maximize the expected return (claims) and, at the same time, to maximize the risk hedge to the reinsurance company. Typically, the risk measures are computed using either the value at risk (VaR) or tail-value at risk (TVar) (Cai et al., 2008). In this work, we are going to use VaR as a metric of risk.



Solution	Layer	Limit	Deductible	Placement	Max Recovery	Premium
1	Layer 1	200	300	60%	120	$40
	Layer 2	200	100	20%	40	$40
2	Layer 1	200	300	30%	60	$25
	Layer 2	200	100	95%	190	$100

Figure 2: Layer example

The problem is then to select the amount of shares or placements on each layer. Due to market constraints, shares are discretized in terms of percentages between 0 and 1. For instance, a contract discretized by 10% leads to a set of eleven possible shares as follows: $S = \{0, 0.1, 0.2, 0.3, ..., 1\}$. If we increase the precision to 5%, the possibilities grow to a set of 21 shares. In other words, the bigger the accuracy, the bigger the search space.

An estimative of solving this kind of problem using an enumerative approach is shown in (Cortes et al., 2013), in which a problem with 10% of discretization and 7 layers takes more than a week in R language. Thus, problems with many layers and high level of discretization cannot be solved by an enumerative algorithm in reasonable time even using parallel computation. Thus, evolutionary algorithms (EAs) represent an interesting solution for this kind of problem, especially those ones called Multi-Objective Evolutionary Algorithms (MOEAs).

The first work to tackle this problem using a single objective EA was (Cortes et al., 2013). The main drawback of this solution is that the user must know the expected return previously to the execution. Then a multi-objective approach was introduced in (Brown et al., 2014), in which a Pareto-based algorithm was proposed using Population-Based Incremental Learning (MO-PBIL). The MO-PBIL is a sophisticated algorithm; however, some drawbacks can be pointed out, such as: (i) the parallel version increases the numbers of individuals, thus, it is tough to obtain speedup; and, (ii) the algorithm tends to move away from bad solutions "genes", even though, bad and good genes can co-exist in the same individual, forcing the algorithm to go away from possible good solutions. Those MO-PBIL problems were solved in (Cortes and Rau-Chaplin, 2016) gaining in both hypervolume and coverage.

In this work, we propose to solve the reinsurance contract problem using the MOEA called Strength Pareto Evolutionary Algorithm 2 (SPEA2), then the results are compared against the Enhanced MO-PBIL. The remaining of this paper is divided as follows: Section 2 explains the basics of Pareto-based multi-objective optimization and describes shortly the problem being solved; Section 3 presents how the algorithm works; Section 4 shows the results of the SPEA2 simulation and compares the results against de enhanced MO-PBIL; finally, Section 5 presents the conclusions of this work and future work.

2 Multiobjective Basics

A multiobjective optimization problem has to deal with two or more conflicting objective function (Deb, 2001) at the same time. These functions must conflict with each other to build a Pareto frontier, in which there are no solutions better than others; otherwise, the answer to the problem would be only one point in the search space.

Thus, assuming that a solution to a MOP is a vector in a search space X with m elements. A function $f : X \rightarrow Y$ evaluates the quality of a solution mapping it into an objective space. Therefore, a multi-objective problem is defined as presented in Equation 1, where f is a vector of objective functions, m is the dimension of the problem and n the number of objective functions.

$$Max\ y = f(x) = (f_1(x_1, ..., x_m), ..., f_n(x_1, ..., x_m)) \quad (1)$$

In order to determine whether a solution belongs to the Pareto frontier or not, we need the concept of optimality, which state that given two vectors $x, x* \in \Re$ and $x \neq x*$, x dominates $x*$ (denoted by $x \succeq x*$) if $f_i(x)$ is not worse than $f_i(x*), \forall\ i$ and $\exists$ at least one i where $f_i(x) > f_i(x*)$ in maximization cases and $f_i(x) < f_i(x*)$ otherwise. Hence, a solution x is said Pareto optimal if there is no solution that dominates x, in such case, x is called non-dominated solution. Mathematically, assuming a set of non-dominated solutions $\wp$, a Pareto frontier(pf) is represented as $pf = \{f_i(x) \in \Re | x \in \wp\}$

2.1 The RCO Problem

The reinsurance process consists of hedging risk from the insurance company to a bigger one, called reinsurance company. The main purpose of doing so is to survive in case of massive claims mainly caused by natural catastrophes. The reinsurance contract optimization problem consists of given a treaty structure to figure out the best combination of placements or shares in order to transfer the maximum of risk, and at the same time, to receive the maximum return when facing massive claims. Therefore, the main purpose of a RCO problem is to find out the best combination of shares or placements which maximize both the transfered risk and the expected return.

The Equation 2 represents the RCO in terms of an optimization problem, where VaR is a risk metric, $\mathbf{R}$ is a function based on a combination of shares pi, and E is the expected value. For further details about the problem refer to (Cortes et al., 2013) and (Cai et al., 2008).

$$\begin{array}{ll} maximize & f_1(x) = VaR_\alpha(\mathbf{R}(\pi)) \\ maximize & f_2(x) = E[\mathbf{R}(\pi)] \end{array} \quad (2)$$

3 SPEA2

The first version of the SPEA for finding approximations of the optimal Pareto set was proposed by Zitzler and Thiele (1998) in a report in 1998, then published in 1999 (Zitzler and Thiele, 1999). At that time, the new algorithm presented good results in comparison against other multiobjective evolutionary algorithms. The new version, called SPEA2 (Zitzler et al., 2001), was proposed in 2001 by the same authors and some modifications were added. In order to save memory, the size of the archive containing the Pareto set is narrowed and

$archive\_size \leftarrow input\_size(n)$
$archive \leftarrow \emptyset$
$pop \leftarrow init\_population(fun_k, pop\_size, dim)$
$archive < -non\_dominated\_sol(pop)$
for (i=1 to max_it) **do**
  $Rt \leftarrow mix(pop, archive)$
  $s \leftarrow compute\_s(Rt)$
  $raw \leftarrow compute\_raw(s)$
  $d \leftarrow compute\_density(Rt)$
  $fitness \leftarrow raw + d$
  $indexes \leftarrow (fitness < 1)$
  $archive\_tmp \leftarrow Rt[indexes,]$
  if $(\#non\_dom == archive\_size)$ **then**
    $archive \leftarrow archive\_tmp$
  else if $(\#non\_dom < archive\_size)$ **then**
    $archive \leftarrow archive\_tmp$
    $archive \leftarrow fill()$
  else
    $archive \leftarrow clustering()$
  end if
  $pop \leftarrow gen\_operators()$
end for

Figure 3: SPEA2 Algorithm

it is always filled up whether there are enough non-dominated points or not. Aiming to fill up the archive using dominated points, the selection process was modified in order to classify the importance of dominated solutions; therefore, the most significant points are chosen. The Figure 3 outlines how the SPEA2 works.

Considering the algorithm, the SPEA2 starts initializing an empty archive of size n and a random population for k objective functions ($k \geq 2$). Then the first set of non-dominated solutions is determined and assigned to the variable *archive*. After that, a temporary population called Rt is made by combining the current population and archive. In fact this combination will have an effect on the algorithms only from the second iteration forward because both tend to be different after that. Afterwards, the algorithm needs to compute the s vector according to Equation 3, which means the cardinality of a point i in terms of dominance, *i.e.*, the cardinality represents how many solutions i dominates in Rt.

$$s(i) = |\{j | j \in Rt \wedge i \succ j\}| \tag{3}$$

The next step is to calculate the raw vector which represents the strengths of each dominator as presented in Equation 4. In other words, if a solution i is dominated by solutions j_1, j_2 and j_3, its raw would be $raw = s[j_1]+s[j_2]+s[j_3]$. Therefore, all non-dominated solutions present $raw = 0$. On the other hand, a high $raw[i]$ value means to be dominated by many individuals.

$$raw[i] = \sum_{j \in Rt, j \prec i} s(j) \tag{4}$$

Although the raw fitness assignment provides a sort of niching mechanism based on the concept of Pareto dominance, it may fail when most individuals do not dominate each other. Therefore, additional density information is incorporated to discriminate between individuals having identical raw fitness values (Zitzler et al., 2001) as shown in Equation 5, where σ is the distance to the k^{th} individual, where the population has been ordered by their Euclidean distance. Usually, k is calculated according to $k = |\sqrt{pop\_size}|$.

$$d(i) = \frac{1}{\sigma_i^k + 2} \tag{5}$$

Thus, the final Rt fitness is equivalent to $fitness = raw + d$. Being $d < 1$ and $raw = 0$ in all non-dominated solutions, therefore all non-dominated solutions will present $fitness < 1$. Then a temporary archive is formed considering only non-dominated solutions. If the number of non-dominated points is equal to the archive size, then the temporary archive replaces the old one. If the length of the temporary archive is less than the archive size, the new archive has to be filled up with non-dominated solutions chosen from the smallest ordered fitnesses. Otherwise, the non-dominated solutions have to be clustered, where the closer the solutions, the bigger the probabilities of being eliminated, thus maintaining the diversity within the archive.

Finally, the population undergoes genetic operators, aiming to move along the search space. Typically, these operators are crossover and mutation, and therefore the same ones originated from genetic algorithms can be applied.

4 Computer Simulations

4.1 Experiment Setup

The experiment was conducted in a Cento OS Linux 6.7, 64 bits, with 2 Intel Xeon ES-2650 processor 2 GHz, each one with 8 cores and hyper-threading, and 256 GB of RAM. We present the Pareto frontier and use the hypervolume as a metric. The code was developed in R 64 bits version 3.3.3. We use 30 trials in order to use statistical parametric tests such as a *t-test* (Montgomery and Runger, 2006).

Regarding the SPEA2, the evaluation metric is the hypervolume presented in Equation 6, which is the area of the dominated points in the Pareto frontier; therefore, the higher the hypervolume, the better the Pareto frontier.

$$hv = volume(\bigcup_{i=1}^{|Q|} v_i) \tag{6}$$

In the SPEA2 algorithm, use used the following operators: selection - tournament between 4 individuals, simple crossover in all selected individuals and probability of mutation equals to 5%; the archive size is equal to 100, and the population size is equal to 50.

4.2 A real data experiment

The first experiment involves a real contract with 7 layers of an anonymized dataset. Figure 4 shows the final Pareto frontier with 7 layers using 100, 250, and 500 iterations. As we can notice, visually the differences between iterations are not remarkable. However, Figure 5 shows that the hypervolume increases as we increase the iteration count, especially when we change from 100 iterations to 250 iterations. The difference is not so impressive if we compare the mean hypervolume between 250 and 500 iterations. Indeed, a bi-caudal t-test with 95% of significance shows that the differences between 100 and 250 iterations in the hypervolume are meaningful ($t = 6.507$). On the other hand, a t-test between 250 and 500 iterations ($t = 1.388$) it is not significant. In this context, if we zoom in some part of the Pareto Frontier as depicted in Figure 6, we can see that the difference exists.

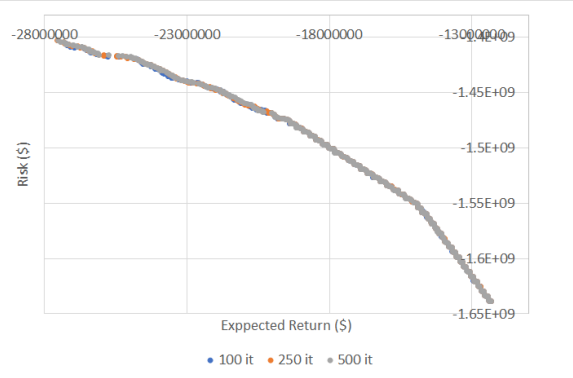


Figure 4: Pareto frontier with 7 layers: 100, 250, and 500 iterations

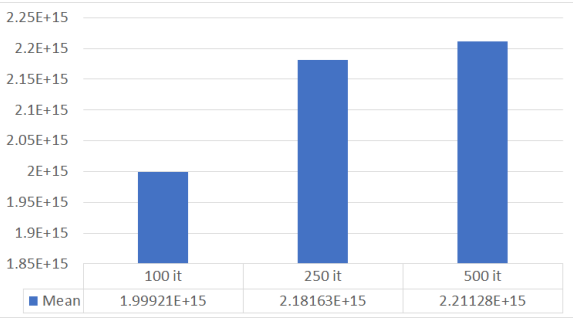


Figure 5: Mean hypervolume as we increase the iteration count

Regarding time, Figure 7 presents the execution time as we increase the iteration count. As

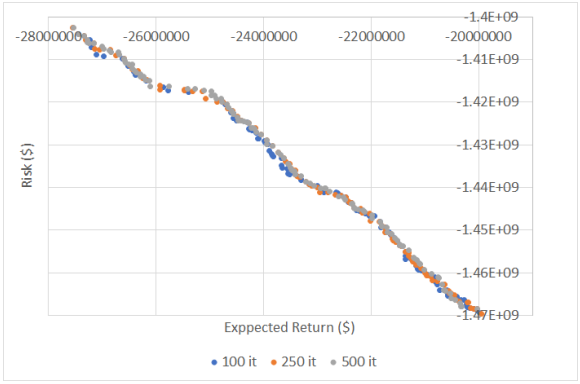


Figure 6: Zooming in an excerpt of the Pareto frontier

we can see, the progression as the number of iterations increase is almost linear.

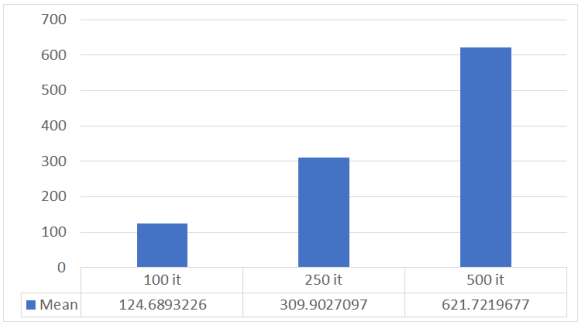


Figure 7: Execution time with 100, 250, and 500 iterations

A comparison between the Pareto frontiers of the SPEA2 and the E-MOPBIL (Cortes and Rau-Chaplin, 2016) with 250 and 500 iterations, respectively, is presented in Figure 8. Visually, we can notice that SPEA2 produces a better Pareto frontier. In order to assure that SPEA2 presents better results, Table 1 shows the hypervolume and the number of solutions on each frontier. In both cases, the hypervolume is better for SPEA2. The number of solutions on the Pareto frontier is also higher in the SPEA2. A bi-caudal t-test with 95% of confidence ($\alpha = 0.05$) indicates that the differences between hypervolumes are meaningful.

4.3 Pushing the Envelope

In order to explore the limits of the SPEA2, we execute the algorithm using 15 and a discretization of 5%. The data was created expanding that one from the real case (7 layers). Because using more layers is computationally intensive, we leverage the situation using parallel computing. To assess the results we used the orthodox weak speedup (Luque and Alba, 2013), in which the same code is executed regardless the number of threads, as shown in Equation 7, where T_1 represent the execution time using one core or thread,

Figure 8: Comparison between SPEA2 and E-MOPBIL with 250 and 500 iterations

Table 1: Hypervolume and number of solutions: SPEA2 vs E-MOPBIL

250 iterations			
	Hypervolume	#Solutions	t
SPEA2	**2.1816E+15**	**286**	7.7
Std. Dev.	8.3141E+13		
E-MOPBIL	1.79E+15	218	
Std. Dev.	2.66E+14		
500 iterations			
	Hypervolume	#Solutions	t
SPEA2	**2.21E+15**	**310**	7.57
Std. Dev.	8.22E+13		
E-MOPBIL	1.901E+15	213	
Std. Dev.	2.0305E+14		

and Tp is the time using p cores or threads.

$$Sp = \frac{T_1}{Tp} \tag{7}$$

The parallelization have been done using the master-slave model in which slaves assume some tasks. In our case, the slaves compute the evaluation functions, which use matrices of dimension of 50,000 x 30. Also, we used the *parallel* package from R that allows us to call the instruction *parApply(cluster, margin, fun)* that executes a function (*fun*) in a matrix by row (margin = 1) in a *cluster* made by 1 or more execution processes (cores). An important detail is to create the cluster using the function *makeCluster(number_cores, "FORK")*. The "FORK" parameter indicates to start threads using the fork model, which is faster in multi-cores architectures than the parameter "SOCK"; however the "FORK" parameter is available only on Unix-like platform. Then, we use the R function *parApply()*, in which the tasks are automatically distributed between the participant cores (threads) created previously using the *makeCluster()* as follows:

```
ncores = number_of_cores
clus = makeClusetr(ncores, "FORK")
...
parApply(clus, 1, function)
...
```

We tested the algorithm using 2, 4, and 8 processes with 15 layers. In fact, we also tried 16 processes; however, the performance was poor due to the excess of fork instruction. Table 2 shows the execution time, speedup, and efficiency using 15 layers as we increase the number of cores with 100, 250, and 500 iterations. As we can see, the best trade-off is obtained using 4 cores with 100 and 500 iterations, even though the efficiency is not impressive we still saving time. Regarding 250 iterations, the best speedup is 1.85; nevertheless, the best trade-off is achieved using 4 cores because the time is similar but uses less computational resources.

Table 2: Time and speedup as we increase the number of cores

100 iterations				
	1 core	2 cores	4 cores	8 cores
Time (s)	203.69	135.85	**129.45**	134.77
Speedup	-	1.50	**1.57**	1.51
Efficiency	-	0.75	0.39	0.19
250 iterations				
	1 core	2 cores	4 cores	8 cores
Time (s)	508.36	347.90	293.04	**274.95**
Speedup	-	1.46	1.73	**1.85**
Efficiency	-	0.73	0.43	0.23
500 iterations				
	1 core	2 cores	4 cores	8 cores
Time	1017.45	698.80	**557.50**	576.01
Speedup	-	1.46	**1.83**	1.77
Efficiency	-	0.73	0.46	0.22

Because the parallelization has been done using the master-slave model the quality of solutions is not affected. Therefore, it is critical to guarantee that the evaluation functions perform intense computational tasks. In this context, we conducted a new experiment to verify if a bigger population increases the speedup. Thus, we execute the application with a population size equals to 100 as depicted in Table 3. As we can notice in the referred table, the speed up increases to 2.44 using eight cores and 500 iterations. Hence, the speedup tends to increase as we expand the population size and the number of iterations.

Table 3: Time and speedup as we increase the number of cores with population size = 100 and layers = 15

100 iterations				
	1 core	2 cores	4 cores	8 cores
Time (s)	389.48	257.00	207.02	183.22
Speedup	-	1.52	1.88	2.13
Efficiency	-	0.76	0.47	0.27
250 iterations				
	1 core	2 cores	4 cores	8 cores
Time (s)	979.51	665.64	527.30	450.04
Speedup	-	1.47	1.86	2.18
Efficiency	-	0.74	0.46	0.27
500 iterations				
	1 core	2 cores	4 cores	8 cores
Time (s)	2064.52	1279.69	932.89	846.58
Speedup	-	1.61	2.21	**2.44**
Efficiency	-	0.81	0.55	0.30

Knowing that the larger the task, the better the speedup, we conducted an experiment using a population size equals to 100 and 30 layers as presented in Table 4 . Due to time constraints, we executed the SPEA 2 only with 100 and 250 iterations.

Table 4: Time and speedup as we increase the number of cores with population size = 100 and layers = 30

100 iterations				
	1 core	2 cores	4 cores	8 cores
Time (s)	933.19	580.17	354.59	289.67
Speedup	-	1.61	2.63	**3.22**
Efficiency	-	0.80	0.66	0.40
250 iterations				
	1 core	2 cores	4 cores	8 cores
Time (s)	2206.85	1510.55	1031.40	866.54
Speedup	-	1.46	2.14	2.55
Efficiency	-	0.73	0.53	0.32

As we can see in Table 4, increasing the number of layers improves the speedup because the granularity of the task is bigger as well, reaching a speedup of 3.22 using eight cores and 100 iterations.

5 Conclusions

This paper presented a new application of the multiobjective algorithm called SPEA2. Results showed that SPEA2 produces better Pareto frontiers as we increase the iteration count. A comparison between E-MOPBIL and SPEA2 was presented as well, indicating that SPEA2 can discover better frontiers than the E-MOPBIL. Also, a parallel version using the master-slave model showed that a good trade-off between speedup and computational resources could be achieved with four cores and a population size equals to 50 with seven layers; and, a reasonable trade-off using eight cores, a population size of 100, and 15 layers. Thus, if we increase the population size and the iterations count, we tend to increase the speedup, as well.

Future work includes implementing new multiobjective algorithms such as NSGA-III (Deb and Jain, 2014) and MOEA/D (Zhang and Li, 2007), hybridizing multiobjective algorithms for solving larger contracts, and parallelization using General Purpose GPUs.

Acknowledgment

We would like to thank CNPq, IFMA, and the Faculty of Computer Science at Dalhousie University who made this work possible.

References

Brown, L., Beria, A. A., Cortes, O., Rau-Chaplin, A., Wilson, D., Burke, N. and Gaiser-Porter, J. (2014). Parallel MO-PBIL: Computing pareto optimal frontiers efficiently with applications in reinsurance analytics, *Conference on High Performance Computing Simulation (HPCS), 2014 International*, pp. 766–775.

Cai, J., Tan, K. S., Weng, C. and Zhang, Y. (2008). Optimal reinsurance under VaR and CTE risk measures, *Insurance: Mathematics and Economics* pp. 185–196.

Cortes, O. A. C. and Rau-Chaplin, A. (2016). Enhanced multiobjective population-based incremental learning with applications in risk treaty optimization, *Evolutionary Intelligence* **9**: 153–165.

Cortes, O. A. C., Rau-Chaplin, A., Wilson, D., Cook, I. and Gaiser-Porter, J. (2013). Efficient optimization of reinsurance contracts using discretized PBIL, *The Third International Conference on Data Analytics*, pp. 18–24.

Deb, K. (2001). *Multi-objective Optimization using Evolutionary Algorithms*.

Deb, K. and Jain, H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based non-dominated sorting approach, part i: solving problems with box constraints, *IEEE Transaction on Evolutionary Computation* **18**(4): 577–601.

Luque, G. and Alba, E. (2013). *Parallel Genetic Algorithms: Theory and Real World Applications*, Springer Publishing Company, Incorporated.

Montgomery, D. C. and Runger, G. C. (2006). *Applied Statistics and Probability for Engineers*, 4 edn.

Zhang, Q. and Li, H. (2007). Moea/d: A multiobjective evolutionary algorithm based on decomposition, *IEEE Transactions on Evolutionary Computation* **11**: 712–731.

Zitzler, E., Laumanns, M. and Thiele, L. (2001). Spea2: Improving the strength pareto evolutionary algorithm, *Technical report*, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich,.

Zitzler, E. and Thiele, L. (1998). An evolutionary algorithm for multiobjective optimization: The strength pareto approach, *Technical report*, Technical Report 43, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich,.

Zitzler, E. and Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach, *IEEE Transactions on Evolutionary Computation* **3**(4): 257–271.