Journal of Polish Safety and Reliability Association
Summer Safety and Reliability Seminars, Volume 6, Number 1, 2015

Guze Sambor
Maritime University, Gdynia, Poland

Numerical application of the SPEA algorithm to reliability multi-objective optimization

Keywords

multi-objective optimization, reliability, 0-1 knapsack problem, SPEA

Abstract

The main aim of the paper is the computer-aided multi-objective reliability optimization using the SPEA algorithm. This algorithm and the binary knapsack problem are described. Furthermore, the computer program that solves the knapsack problem with accordance to SPEA algorithm is introduced. Example of the possible application of this program to the multi-objective reliability optimization of exemplary parallel-series system is shown.

1. Introduction

The technological development requires the use of more advanced methods and techniques to solve the engineering problems. This is a result of the fact, that the technical systems are becoming more complex. Thus, the problems of designing optimal systems or finding optimal solutions are met in many areas of present science, technology and economics. When we take into account the optimization problem, the three elements need to be specify: a model of the phenomenon of distinguished decision variables, objective functions also known as a quality criterion and constraints [2], [17]-[18]. This is a classical point of view on optimization problem. With the reference to the current state-of-the-art in the reliability and safety analysis of the technical systems the increasing of their complexity are noted [4], [6]-[7]. This implies that the improvement of the system [5], [8]-[9] only in one direction is no longer sufficient. Therefore, the one-objective optimization [3], [5]-[7], [12], [14], [16] should be replaced by multi-objective approach [2], [9], [14],[16]-[19].

Most of the presented results take into account only one criterion for the optimization. There are the known methods to the reliability prediction and optimization of complex technical systems related to their operation processes, where the time is a fundamental criterion [3]-[7], [10]. The tools for solving the problems of complex technical systems availability, safety and cost optimization [3]-[7] are also introduced. All of these problems can be solved by well-known deterministic optimization methods for engineering and management [12], [14], [16]. These problems are important according to the critical infrastructures analysis and modelling [1], [13], too. Because that applies to everyday human activities, the multi-objective approach to the improvement operation process, reliability and safety need to be used. Thus, the proposition of transformation a reliability optimization problem to the binary knapsack problem [2], [11], [17]-[19] is presented in the paper. Furthermore, a possible application of the computer program to the multi-criteria reliability optimization of the technical system is shown. This implements the Strength Pareto Evolutionary Algorithm [15], [17]-[19], which is recognized as one of the most effective [17].

2. Concepts of the single and multi-objected optimization

The basic aim of both approaches to optimization is to get the solution for minimizing or maximizing problem. The number of the objective functions is a fundamental difference. Thus, the definition of the single-objected optimization problem is following:

$$F(x_i) \to \min \text{ or } F(x_i) \to \max,$$

$$l_j(x_i) \le 0, l_j(x_i) \le 0, x_i \ge 0, i, j = 1,2,...,n \quad (1)$$

where

x_i - decision variables, $i = 1,2,...,n$;
$F(x_i)$ - goal(objective) function;
$l_j(x_i)$ - limits function (low or high) for decision variables, $i, j = 1,2,...,n$.

The solution for above problem is to find the unknown goal function.

In the other hand, the multi-objective optimization model can be described as a vector function f that maps a tuple of m decision variables (parameters) to a tuple of n objectives functions, and a set of k constrains. Objective functions and constraints are functions of the decision variables. The formal notation is as follows [2], [17]-[18]:

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x})) \to \text{max or min}$$
$$\text{subject to } \boldsymbol{e}(\boldsymbol{x}) = (e_1(\boldsymbol{x}), e_2(\boldsymbol{x}), \ldots, e_k(\boldsymbol{x})) \le \boldsymbol{0} \text{ or}$$
$$\boldsymbol{e}(\boldsymbol{x}) = (e_1(\boldsymbol{x}), e_2(\boldsymbol{x}), \ldots, e_k(\boldsymbol{x})) \ge \boldsymbol{0}, \qquad (2)$$

where

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_m) \in \boldsymbol{X},$$
$$\boldsymbol{y} = (y_1, y_2, \ldots, y_n) \in \boldsymbol{Y},$$

and $\boldsymbol{x}$ is the decision vector, $\boldsymbol{y}$ is the objective vector, $\boldsymbol{X}$ is denoted as the decision space and $\boldsymbol{Y}$ is called the objective space.
The constraints $\boldsymbol{e}(\boldsymbol{x}) \le \boldsymbol{0}$ ($\boldsymbol{e}(\boldsymbol{x}) \ge \boldsymbol{0}$) is described the set of feasible solution for maximization (minimization) problems.

The set

$$\boldsymbol{X}_f = \{\boldsymbol{x} \in \boldsymbol{X} \mid \boldsymbol{e}(\boldsymbol{x}) \le \boldsymbol{0}\}$$
$$(\boldsymbol{X}_f = \{\boldsymbol{x} \in \boldsymbol{X} \mid \boldsymbol{e}(\boldsymbol{x}) \ge \boldsymbol{0}\}) \qquad (3)$$

of decision vectors $\boldsymbol{x}$ that satisfy the constraints $\boldsymbol{e}(\boldsymbol{x}) \le \boldsymbol{0}$ ($\boldsymbol{e}(\boldsymbol{x}) \ge \boldsymbol{0}$) is called the feasible set for maximization (minimization) problems. Following the above, its image, i.e., the feasible region in the objective space, is denoted as

$$\boldsymbol{Y}_f = \boldsymbol{f}(\boldsymbol{X}_f) = \bigcup_{x \in \boldsymbol{X}_f} \{\boldsymbol{f}(\boldsymbol{x})\}. \qquad (4)$$

2.1. Introduction to Pareto-optimality

According to above notation, there exist the set of multi-objective optimization problem solutions. It consists of all decision vectors for which the corresponding objective vectors cannot be improved in any dimension without degradation in another. They are called Pareto optimal (Pareto frontier/ Pareto set/ Pareto front), what is related to the concept of domination vector by vector. It is simple to explain based on following *Definitions 1-4*, [17]-[18].

Definition 1. Let us take into account a maximization (minimization) problem and consider two decision vectors $\boldsymbol{a}, \boldsymbol{b} \in X$, then $\boldsymbol{a}$ is said to dominate $\boldsymbol{b}$ ($\boldsymbol{a} \succ \boldsymbol{b}$ or $\boldsymbol{a} \prec \boldsymbol{b}$) if and only if

$$\forall i \in \{1,2,...,n\}: f_i(\boldsymbol{a}) > f_i(\boldsymbol{b}) \; (f_i(\boldsymbol{a}) < f_i(\boldsymbol{b}))$$
$$\wedge$$
$$\exists j \in \{1,2,...,n\}: f_j(\boldsymbol{a}) > f_j(\boldsymbol{b}) \; (f_j(\boldsymbol{a}) < f_j(\boldsymbol{b}).) \qquad (5)$$

Definition 2. Let us take into account a maximization (minimization) problem and consider two decision vectors $\boldsymbol{a}, \boldsymbol{b} \in X$, then $\boldsymbol{a}$ is said to weak dominate $\boldsymbol{b}$ if and only if

$$\forall i \in \{1,2,...,n\}: f_i(\boldsymbol{a}) \ge f_i(\boldsymbol{b}) \; (f_i(\boldsymbol{a}) \le f_i(\boldsymbol{b}))$$
$$\wedge$$
$$\exists j \in \{1,2,...,n\}: f_j(\boldsymbol{a}) \ge f_j(\boldsymbol{b}) \; (f_j(\boldsymbol{a}) \le f_j(\boldsymbol{b}).) \qquad (6)$$

Definition 3. Let us take into account a maximization (minimization) problem and consider two decision vectors $\boldsymbol{a}, \boldsymbol{b} \in X$, then $\boldsymbol{a}$ is said to be indifferent to $\boldsymbol{b}$ if and only if

$$\forall i \in \{1,2,...,n\}: f_i(\boldsymbol{a}) not \ge f_i(\boldsymbol{b}) \wedge f_i(\boldsymbol{b}) not \ge f_i(\boldsymbol{a})$$
$$(f_i(\boldsymbol{a}) not \le f_i(\boldsymbol{b}) \wedge f_i(\boldsymbol{b}) not \le f_i(\boldsymbol{a})) \qquad (7)$$

The graphical interpretation of the above definition are presented in *Figure 1*.

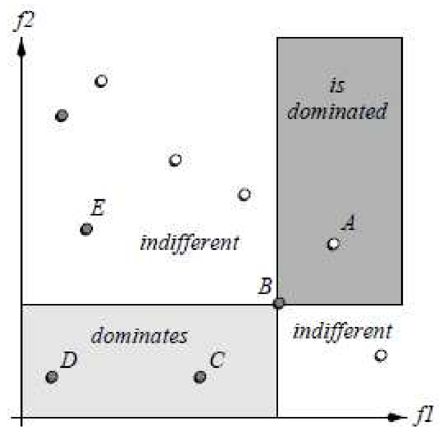

Figure 1. Possible relation in objective space [17]

According to given relation between the solutions in objective space (*Definitions 1-3*), it is possible to define the Pareto optimality. However, the key issue is specifying the concept of non-dominated decision vector [17]-[18].

Definition 4. A decision vector $\boldsymbol{x} \in \boldsymbol{X}_f$ is said to be non-dominated regarding to set $A \subseteq \boldsymbol{X}_f$ if and only if

$$not\exists_{\boldsymbol{a}\in A}\boldsymbol{a} \succ \boldsymbol{x}. \quad (8)$$

It means, that all decision vectors which are not dominated by another decision vector are called non-dominated. Moreover, the Pareto optimality is defined as follows [17].

Definition 5. A decision vector $\boldsymbol{x}$ is said to be Pareto optimal if and only if $\boldsymbol{x}$ is non-dominated regarding $\boldsymbol{X}_f$.

In the other words, when the decision vectors are non-dominated within the entire search space, they are denoted as Pareto optimal or efficient. Its graphical representation is called Pareto-optimal front or surface (see *Figure 2*).

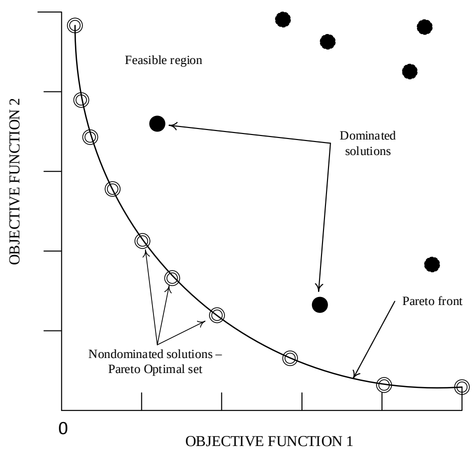

Figure 2. Illustrative example of Pareto-optimal front for minimizing problem

Moreover, when a set of choices and a way of valuing them are given, the Pareto front is the set of choices that are Pareto optimal (efficient). Regarding to the set of choices that are Pareto-optimal a decision maker can make tradeoffs within this set, in place of consideration the full range of every parameter. It means that the shape of the Pareto front indicates the nature of the trade-off between the different objective functions.
In language of the statistical decision theory the above approach can be compare to an admissible decision rule. It is a rule for making a decision such that there is not any other rule that is always "better" than it. In general, the set of admissible rules for most decision problems is large, sometimes infinite. Therefore, this is not a sufficient criterion to take into account a single rule, but should favor admissible rules. The Pareto-optimality gives a suggestion what decision maker can consider as optimal (maximal or minimal).

2.2. Methods and algorithms for multi-objective optimization

The most frequently used multi-objective analytical deterministic or non-deterministic optimization methods are as follows:
- Weighted Objective Methods;
- Hierarchical Optimization Method;
- Trade-Off Method;
- Global Criterion Method;
- Method of Distance Functions;
- Min-Max Methods;
- Goal Programming Method.

The above approaches can provide general tools for solving optimization problems to obtain a global or an approximately global optimum. In the second case the better way to work out is using the evolutionary or genetic algorithms, such as:
- Strength Pareto Evolutionary Algorithm (SPEA);
- VEGA – Vector Evaluated Genetic Algortihm;
- HLGA - Hajela and Lin's Weighting-based Genetic Algorithm;
- NPGA – Niched Pareto Genetic Algorithm.

General operation of genetic or evolutionary algorithms is based on the following steps (see *Figure 3*):
1. Initialization.
2. Calculate fitness.
3. Selection/Recombination/Mutations (parents and children).
4. Finished.

The simplified drawing showing the appearance of the basic genetic algorithm is presented in *Figure 3*.
The data is represented by population of chromosomes, where each of them is composed of a string of bits (see *Figure 3*).

In the paper the Strength Pareto Evolutionary Algorithm (SPEA) and its numerical realization [2], [17]-[18] is considered as a representative evolutionary algorithm. The basic notations for correct presentation of it are as follows:
t - number of generation,

P_t - population in generation *t*,
$\overline{P}_t$ - external set in generation *t*,
$\overline{P}'$ - temporary external set,
P' - temporary population.

Additionally, the following input parameters are given:
N - population size,
$\overline{N}$ - maximum size of external set,
T - maximum number of generations,
p_c - crossing probability,
p_m - mutation probability,
A- set of non-dominated solutions.

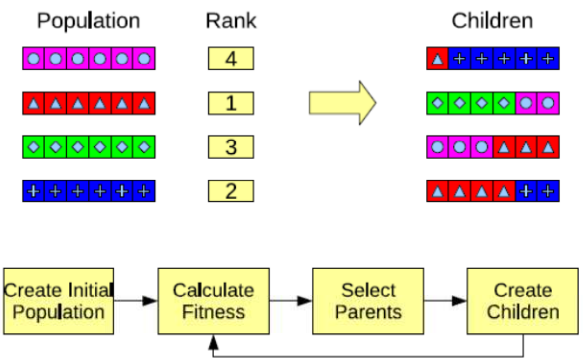

Figure 3. Basic genetic/evolutionary algorithm [16]

The Strength Pareto Evolutionary Algorithm [2], [15]:

<u>*Step 1*. Initialization:</u>
The initial population P_0 is generated according to procedure:

a) To get item *i*.
b) To add item *i* to set P_0.

Next, the empty external set $\overline{P}_0$ is generated, where *t = 0.*

<u>*Step 2*. The complement of the external set is done.</u>
Let $\overline{P}' = \overline{P}_t$

a) To copy non-dominated items from population P_t to population $\overline{P}'$.
b) To remove dominated items from set $\overline{P}'$.
c) To reduce the cardinality of the set $\overline{P}'$ to value $\overline{N}$, using clustering and the solution give into $\overline{P}_{t+1}$.

<u>*Step 3*. Determination fit function.</u>

The value of the fit function F for items from sets P_t and $\overline{P}_t$ can be found according to following procedure:
The real value $S \in [0,1)$ is assigned for every item $i \in \overline{P}_t$ (called power). This value is proportional to number of items $j \in P_t$, which represents the solutions dominated by item *i*.
The adaptation of item *j* is calculated as sum of all items from external set, represents solution dominated by item *j*, increased by 1.
The aim of addition 1 is to ensure that items $i \in \overline{P}_t$ will have better value of fit function than items from set P_t, i.e.

$$S(i) = \frac{n}{N+1}, \qquad (9)$$

where:
$S(i)$ - power of item *i*,
n - number of items in population dominated by item *i*.

It is assumed that value of fit function for item *i* is equal to his power, i.e.

$$F(i) = S(i). \qquad (10)$$

<u>*Step 4*. Selection</u>
Let P' = Ø.
For i = 1,2,… k do

a) To choose randomly two items $i, j \in P_t \cup \overline{P}_t$.
b) If $F(i) < F(j)$ then $P' = P' \cup \{i\}$ else $P' = P' \cup \{j\}$, under assumption that value of fit is minimizing.

<u>*Step 5*. Recombination</u>.
Let P''=Ø.
For i = 1,2,…N/2 do:

a) To choose two items $i, j \in P'$ and to remove it from $\overline{P}'$.
b) To create items: k, l by crossing the items i, j.
c) To add items k, l to set P'' with probability p_c, else add items i, j to set P''.

<u>*Step 6*. Mutation</u>
Let P'''=Ø.
For every item $i \in P''$ do:

a) To create item *j* by mutation the item *i* with probability p_m.

b) To add item j to set P'''.

<u>*Step 7.* Finished</u>
Let $P_{t+1} = P'''$ and $t = t + 1$. If $t \geq T$ then return A – non-dominated solution from population P_t and finish else back to *Step 2*.

The graphical representation of the above algorithm's steps is shown in *Figure 4*.

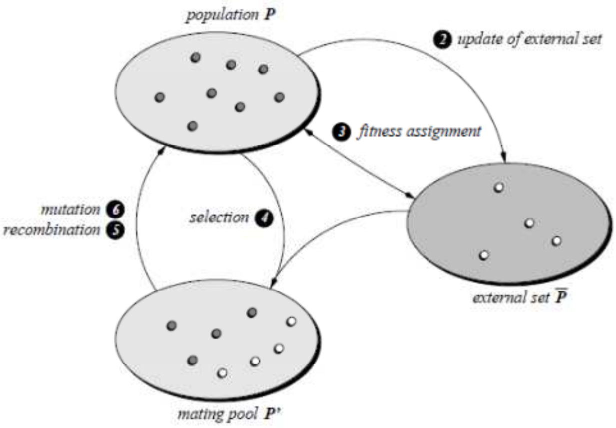

Figure 4. General steps in the SPEA [17]

3. The knapsack problem

The knapsack problem has been known since 1897 as a combinatorial optimization problem. The general description is based on given a set of items, each with a mass and a value. There is determined the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible (according to (1)). The knapsack problem is a subset of NP-hard problems. It means that there is non-polynomial algorithm to solve this problem. Therefore, the knapsack problem has been modified many times. i.e. to form of the 0-1 knapsack problem. This modification allows for formulation of knapsack problem as multi-objective optimization problem.

3.1. The 0-1 knapsack problem – basic notations

Generally, a 0-1 knapsack problem consists of a set of items, weight and profit associated with each item, and an upper bound for the capacity of the knapsack. The main goal is to find a subset of items which maximizes the profits and all selected items fit into the knapsack, i.e., the total weight does not exceed the given capacity [2], [11], [17], [18].
This single-objective problem can be extended directly to the multi-objective case by allowing an arbitrary number of knapsacks. Formally, the multi-objective 0-1 knapsack problem can be defined in the following way [2], [17], [18] according to formula (2):
Given a set of m items and a set of n knapsacks, with

$p_{i,j}$ = profit of item j according to knapsack i,
$w_{i,j}$ = weight of item j according to knapsack i,
c_i = capacity of knapsack i,

find a vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_m) \in \{0,1\}^m$, such that

$$\forall i \in \{1,2,...,n\} : e_i(\boldsymbol{x}) = \sum_{j=1}^{m} w_{i,j} \cdot x_j \leq c_i \qquad (11)$$

and for which $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$ is maximum, where

$$f_i(\boldsymbol{x}) = \sum_{j=1}^{m} p_{i,j} \cdot x_j \qquad (12)$$

and $x_j = 1$ if and only if when item j is chosen.

3.2. The 0-1 knapsack problem solutions

The solutions of knapsack problem can be described in terms of a genetic or evolutionary methods. In the paper, the SPEA algorithm from Section 2.2., is proposed to solve the problem. The computer program to find the solution of the 0-1 knapsack problem is implemented in C programming language with using PISA project codes, developed in Computer Engineering and Networks Laboratory of ETH Zurich and available on website http://www.tik.ee.ethz.ch/sop/pisa/?page=pisa.php.
PISA is a text-based interface for search algorithms. It splits an optimization process into two modules. One module, called the Variator**,** contains all parts specific to the optimization problem (e.g., 0-1 knapsack problem). The second module, called the Selector, contains the parts of an optimization process which are independent of the optimization problem (mainly the selection process, i.e. SPEA2). These two modules are implemented as separate programs which communicate through text files as is presented in *Figure 5* [19].

Figure 5. The schema of PISA project components [19]

There are the six text files that are a platform to exchange of data between the Variator (Knapsack) and the Selector (SPEA2). According to documentation of Knapsack module, the most important in common files is PISA_cfg file that consists the following parameters:

- *alpha* - number of individuals in initial population;
- *mu* - number of individuals selected as parents;
- *lambda* - number of offspring individuals;
- *dim* - number of objectives

Unfortunately there are some limitations to the Knapsack module. It works only when *mu* == *lambda*. In the other hand, if an odd number is chosen for *mu* and *lambda*, the last individual in the mating pool (see *Figure 4*) can only undergo mutation, because it has no recombination partner.

Additionally, two files of the parameters for both programs are available.
In case of the Variator the parameters are as follows:

- *seed* - seed for random number generator;
- *length* - length of the binary string (length of the chromosome);
- *maxgen* - maximum number of generations (stop criterion) 5
- *outputfile* – name of file for output of the last population in archive, where one individual is written per line using the following format:
  ID (objective 1) (objective 2) ... (objective dim) bit-vector;
- *mutation_type* – mutation type, where 0 = no mutation, 1 = one bit mutation, 2 = independent bit mutation;
- *recombination_type* – recombination type, where 0 = no recombination, 1 = one point crossover, 2 = uniform crossover;
- *mutation_probability* – probability that individual is mutated;
- *recombination_probability* - probability that two individuals are recombined;
- *bit_turn_probability* - probability, that bit is turned when mutation occurs only used for independent bit mutation.

For the Selector (SPEA2) the following parameters are included:

- *seed* - seed for random number generator;
- *tournament* - parameter for number of the tournament selection.

The computer program is implemented with accordance to formulae (1)-(12).

4. Reliability of the two-state parallel-series system

In the case of two-state reliability analysis of parallel-series systems we assume that [2], [4]:

– n is the number of system components,
– E_{ij}, $i = 1,2,...,k_n$, $j = 1,2,...,l_i$, are components of a system,
– T_{ij} are independent random variables representing the lifetimes of components E_{ij}, $i = 1,2,...,k_n$, $j = 1,2,...,l_i$,
– $R_{ij}(t) = P(T_{ij} > t), t \in < 0, \infty)$, is a reliability function of a component E_{ij}, $i = 1,2,...,k_n$, $j = 1,2,...,l_i$,
– $F_{ij}(t) = 1 - R_{ij}(t) = P(T_{ij} \le t), t \in < 0, \infty)$, is the distribution function of the component E_{ij} lifetime T_{ij}, $i = 1,2,...,k_n$, $j = 1,2,...,l_i$, also called an unreliability function of a component E_{ij}, $i = 1,2,...,k_n$, $j = 1,2,...,l_i$.

Moreover, we assume that components E_{i1}, E_{i2}, …, E_{il_i}, $i = 1,2,...,k_n$, create a parallel subsystem S_i, $i = 1,2,...,k_n$, and that these subsystems create a series system.

Definition 6. A two-state system is called parallel-series if its lifetime T is given by

$$T = \min_{1 \le i \le k_n} \{ \max_{1 \le j \le l_i} T_{ij} \}. \quad (13)$$

According to above definition, the reliability function of the two-state parallel-series system is given by

$$\overline{R}_{k_n, l_1, ..., l_{k_n}} j(t) = \prod_{i=1}^{k_n} \left[1 - \prod_{j=1}^{l_i} F_{ij}(t) \right], t \in (-\infty, \infty). \quad (14)$$

5. 5. Multi-criteria methods for reliability optimization problem

We assume that the two-state parallel-series system with three main units S_i is given ($i = 1,2,3$). Every unit is the parallel subsystem consists of maximum components which can be chosen to provide redundancy (see *Figure 4*). These maximal numbers are equal to:

- 4, for unit S_1;
- 3, for unit S_2;
- 3, for unit S_3.

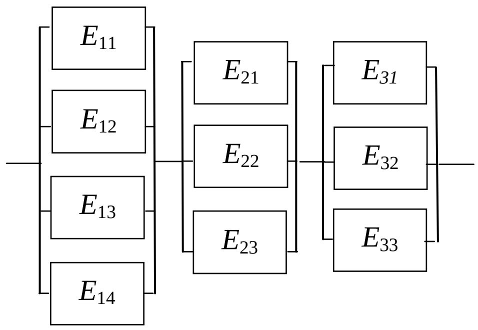

Figure 6. Exemplary scheme of a parallel-series system

Every component of the system can have two states, functioning with the nominal capacity or total failure, corresponding to capacity 0. The main characteristics of these components are lifetime and cost. The exemplary system components are given in *Table 1.*

Table 1. Exemplary characteristics of the system components

Subsystem	Component type	Lifetime [h]	Cost [USD]
1	1	350	9899
	2	840	11259
	3	255	6137
	4	190	4122
2	1	198	3818
	2	740	10016
	3	500	7213
3	1	960	10189
	2	180	4991
	3	607	15683

In real world application, the main problem can be formulated as the question how to create new system or to redesign existing one for extending its time to failure as much as possible with a cost as low as possible. It means that the goal of the problem is to maximize the time to failure of the system and to minimize the cost. This is the classical two-objective optimization. The solution of the problem can be done by a transformation the reliability problem to the 0-1 knapsack problem. This can be done, according to the Section 3, when the assumptions are as follows:

- c_i is the time to failure of designed system;
- $p_{i,j}$ is the profit equal to lifetime of using the particular component;
- $w_{i,j}$ is the cost of the component usage and installation.

Furthermore, let us assume that a chromosome represents the reliability of whole system. In this chromosome the gen equal to 1 means that given component is into knapsack. On the other hand, the gen in this chromosome is equal to 0 means that this component is not in knapsack. The exemplary chromosome of the system is presented in *Figure 5*. The length of a chromosome is the number of the components, which are under investigation (number of system components, see *Figure 7*).

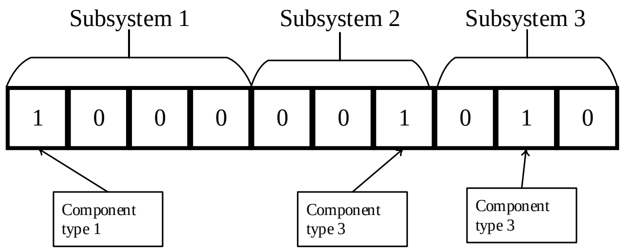

Figure 7. Exemplary chromosome describes the system components

The showcase of the possibility usage of the computer program is presented in *Example.*

Example
Let us take into account the two-state parallel-series system with three main units S_i, where $i = 1,2,3$. Furthermore, the optimization of the time to failure according to minimal cost is needed to done. There is the set of components that can be selected to improve the system reliability.

To solve above problem, the computer program proposed in Section 3.2. is used according to formulae (13) – (14). The five cases are considered for showing the capabilities of this program. In every case five generations of the algorithm (presented in Section 2.2.) are taken into account for program execution.
To use the computer program, the following parameters for four cases commonly (Case 1-4) in configure file are fixed. In the case 5 two parameters are changed.

The input file *"PISA_CFG"* for cases 1-4 is as follows:
- *alpha 50*
- *mu 50*
- *lambda 50*
- *dim 2*

and for case 5 is given as
- *alpha 50*
- *mu 20*
- *lambda 20*
- *dim 2*

The initial population for all considered cases (1 – 5) is given in *Table 2*. It describes the set of components which can be used to improving the system reliability.
The common parameters in the file "*Knapsack_param.txt*" are as follows:

- *length 10*
- *maxgen 5*
- *mutation_probability 0.5*
- *recombination_probability 0.5*
- *bit_turn_probability 0.05.*

Let us consider the following cases.
Case 1:

- *mutation_type 1*
- *recombination_type 1.*

Case 2:

- *mutation_type 1*
- *recombination_type 2.*

Case 3:

- *mutation_type 2*
- *recombination_type 2.*

Case 4:

- *mutation_type 2*
- *recombination_type 1.*

Case 5:

- *mutation_type 2*
- *recombination_type 2*
- *mu 20*
- *lambda 20.*

The mutation type and the recombination type are different in four proposed cases. The last one has the same types of mutation and recombination as case 3, but there are different parameters *mu* and *lambda.*
The results of execution of the program are given in *Tables 3-7* and are presented in *Figures 8-12*.

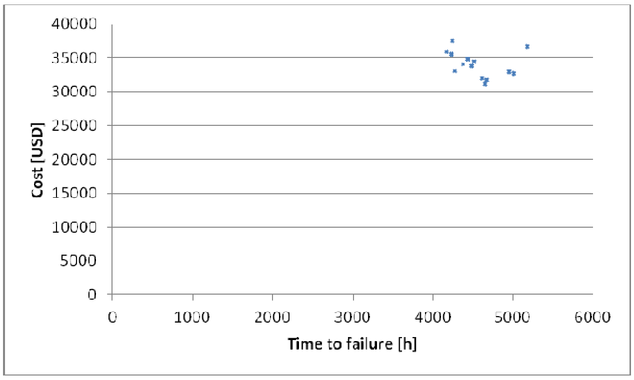

Figure 8. Exemplary results for 5 generations – case 1

Table 2. PISA_INI – initials individuals

ID	Bit string	Time to Failure	Cost [USD]
0	0111000010	5460	42100
1	0011110111	4040	35000
2	0111111101	5010	37900
3	0101101011	4490	31100
4	1011000001	4760	39600
5	1011010111	4010	35900
6	1011000000	5630	46100
7	0101101010	4830	32000
8	1001000011	5260	37800
9	0101000111	4840	32900
10	0110000111	5600	43700
11	0001110110	4910	41500
12	1010011000	4870	39700
13	1111011001	4480	32300
14	1001001110	5470	35100
15	1001111000	4830	38800
16	1001111111	4430	31400
17	0010010011	5020	44000
18	0110010001	5070	44900
19	1101110111	4040	35000
20	1010110101	4550	44900
21	0100101010	5840	40100
22	0110011100	5280	42200
23	1000100110	6090	46800
24	0011010001	4590	42400
25	1011110001	4620	41500
26	0010111010	5260	40400
27	1110011110	5780	40400
28	0011110111	4040	35000
29	1111001101	5590	37600
30	1111110100	4960	42400
31	0101110100	4960	42400
32	0100001000	7470	52200
33	0010110010	4840	44900
34	1100011010	5780	40400
35	0000111000	5840	46900
36	1011101010	4280	32900
37	1000010001	5740	47100
38	1011101000	4860	39400
39	1111011111	4430	31400
40	0100110110	5390	44000
41	0101111110	5300	37900
42	0110101100	5340	41900
43	1110010000	5000	43300
44	0000111000	5840	46900
45	0000011101	5190	38600
46	1001111011	4430	31400
47	0000000100	7830	59600
48	0110010010	5360	44900
49	1100001100	6530	44100

Case 1.

Table 3. Results of the computer program execution, where the grey rows are Pareto-optimal front – case 1

ID	Bit string	Time to Failure [h]	Cost [USD]
1	1011010001	5010	32700
3	0101101011	4170	35900
4	1100111011	4270	33100
6	1111010111	5180	36700
10	1001111111	4230	35600
15	1011010001	5010	32700
16	1001111111	4230	35600
17	1011100001	4950	33000
25	1111010011	5180	36700
27	1001111111	4230	35600
30	1011010001	5010	32700
31	1011100001	4950	33000
32	1001111111	4230	35600
33	1011000111	4480	33900
35	1001101010	4240	37500
36	1011101010	4380	34100
38	1011000111	4480	33900
39	1111011111	4230	35600
40	1111010111	5180	36700
41	0101101011	4170	35900
42	1111111111	4230	35600
43	1111010000	4670	31800
44	1011000110	4440	34800
46	1001111011	4230	35600
47	1001010111	4510	34500
52	1011000011	4480	33900
54	1001111011	4230	35600
55	1011100001	4950	33000
56	1011100101	4950	33000
57	0011101011	4170	35900
59	1111110011	4620	32000
60	0101111111	4230	35600
61	1011000110	4440	34800
62	1011010001	5010	32700
64	1011010001	5010	32700
69	1001111011	4230	35600
72	1011000110	4440	34800
76	1001101010	4240	37500
79	1011010111	4650	31100
82	1011000111	4480	33900
83	1011000110	4440	34800
84	1001110011	4620	32000
88	1111011011	4230	35600
89	1111010111	5180	36700
92	1011000011	4480	33900
94	1101111011	4230	35600
95	1001101011	4170	35900
97	1001111011	4230	35600
98	1011000111	4480	33900
99	1011010011	4650	31100

Case 2.

Table 4. Results of the computer program, where the grey rows are Pareto-optimal front – case 2

ID	Bit string	Time to Failure [h]	Cost [USD]
2	1111100011	5120	37000
3	0101101011	4170	35900
4	1001010011	4510	34500
11	0111100011	5120	37000
13	1001011110	4300	37200
14	1011100010	4660	33000
16	1001111111	4230	35600
18	1011100110	4660	33000
23	0111100011	5120	37000
25	1011100110	4660	33000
28	1011010001	5010	32700
29	1001010011	4510	34500
31	1001011110	4300	37200
33	1001100111	4450	34800
34	1001010011	4510	34500
39	1111011111	4230	35600
40	1011100110	4660	33000
43	1011000110	4440	34800
44	1001100011	4450	34800
45	1011100110	4660	33000
46	1001111011	4230	35600
47	0111010011	5180	36700
50	1011100110	4660	33000
54	0011101011	4170	35900
56	0111100011	5120	37000
57	0111100011	5120	37000
60	1001010011	4510	34500
62	1001010111	4510	34500
63	1111100011	5120	37000
64	0111100011	5120	37000
66	1011100110	4660	33000
68	1111011111	4230	35600
70	1011100110	4660	33000
71	1001011111	4230	35600
74	1001100111	4450	34800
75	1011010010	4720	32700
79	0111100011	5120	37000
80	1001100111	4450	34800
81	1001101111	4170	35900
83	1011010101	5010	32700
84	1001010011	4510	34500
85	1011100010	4660	33000
88	1001010011	4510	34500
90	0111100111	5120	37000
93	1001010011	4510	34500
94	1011010001	5010	32700
95	0001101111	4170	35900
97	1011100110	4660	33000
98	1001011110	4300	37200
99	1001011110	4300	37200

Case 3.

Table 5. Results of the computer program, where the grey rows are Pareto-optimal front – case 3

ID	Bit string	Time to Failure [h]	Cost [USD]
4	1011100110	4660	33000
6	1001101010	4240	37500
7	1011010110	4720	32700
8	1111010111	5180	36700
11	1011101010	4380	34100
12	1011011110	4440	33800
13	1001011110	4300	37200
15	1011010101	5010	32700
18	1011101010	4380	34100
20	1011010110	4720	32700
25	1011101010	4380	34100
26	1001101010	4240	37500
27	1011010110	4720	32700
28	1001011010	4300	37200
29	1011010110	4720	32700
31	1001011010	4300	37200
32	1001011110	4300	37200
33	1011010110	4720	32700
34	1001011110	4300	37200
36	1011101010	4380	34100
43	1011010110	4720	32700
44	1011100110	4660	33000
45	1011010110	4720	32700
47	1011101010	4380	34100
51	1011000111	4480	33900
52	1011101010	4380	34100
53	1011100110	4660	33000
54	1011010110	4720	32700
57	1011101010	4380	34100
58	1011100110	4660	33000
59	1001101010	4240	37500
60	0011101010	4380	34100
61	1011101010	4380	34100
64	1011101010	4380	34100
65	1011010110	4720	32700
69	1011011110	4440	33800
70	1011100110	4660	33000
72	1011000111	4480	33900
73	1001011110	4300	37200
74	1011010010	4720	32700
76	1011101010	4380	34100
77	1001011110	4300	37200
80	1011000111	4480	33900
82	1011101110	4380	34100
83	1011010001	5010	32700
84	1011100110	4660	33000
86	1011100110	4660	33000
94	1011000111	4480	33900
96	1111010111	5180	36700
99	1001011110	4300	37200

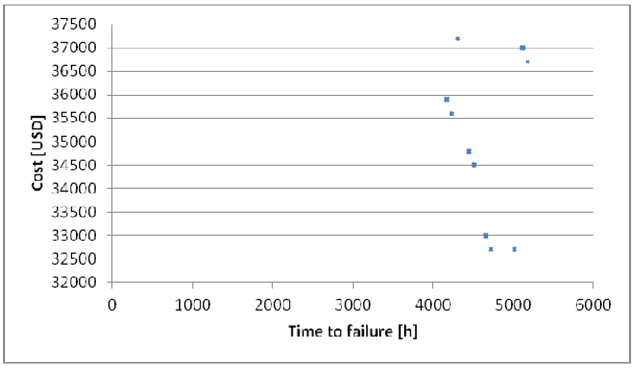

Figure 9. Exemplary results for 5 generations – case 2

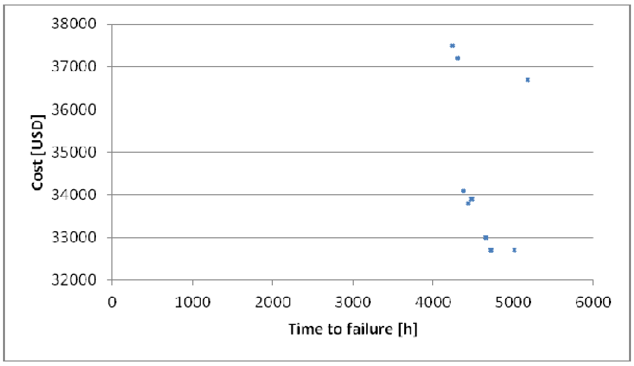

Figure 10. Exemplary results for 5 generations – case 3

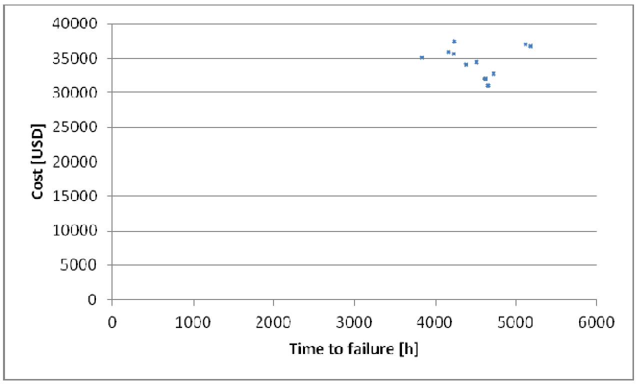

Figure 11. Exemplary results for 5 generations – case 4

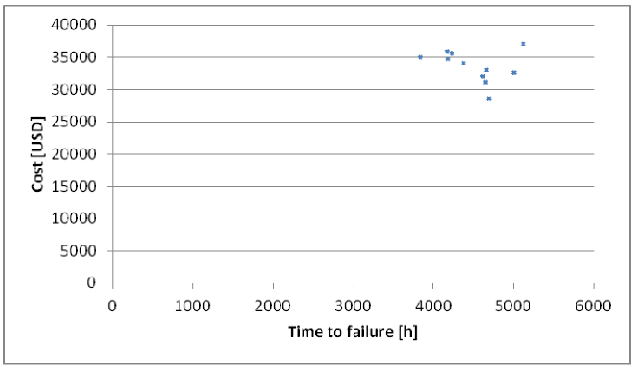

Figure 12. Exemplary results for 5 generations – case 5

Case 4.

Table 6. Results of the computer program, where the grey rows are Pareto-optimal front

ID	Bit string	Time to failure [h]	Cost [USD]
1	0011110111	4620	32000
5	1011010111	4650	31100
6	0101101110	3830	35000
8	1111010111	5180	36700
9	1111010111	5180	36700
10	1011010110	4720	32700
11	0101110111	4620	32000
12	0101110111	4620	32000
13	1001101111	4170	35900
19	1101110111	4620	32000
22	1111010111	5180	36700
28	0011110111	4620	32000
31	1011010111	4650	31100
32	1001010011	4510	34500
33	1101110111	4620	32000
34	0011110111	4620	32000
35	0101110111	4620	32000
36	1011101010	4380	34100
37	0011110111	4620	32000
43	0011110111	4620	32000
44	0011110111	4620	32000
45	1101110111	4620	32000
48	1011110111	4620	32000
49	0011101110	4380	34100
51	1011010111	4650	31100
52	1011101010	4380	34100
53	0111100111	5120	37000
55	0101110111	4620	32000
56	0101110111	4620	32000
59	1011010111	4650	31100
61	1101110111	4620	32000
64	1001110111	4620	32000
65	1011010111	4650	31100
67	1011010111	4650	31100
69	1001101110	4240	37500
70	1111010111	5180	36700
71	1111010111	5180	36700
72	0111110111	4620	32000
74	0001110111	4620	32000
75	0101110111	4620	32000
79	1101110111	4620	32000
80	1111110111	4620	32000
81	1001101110	4240	37500
82	1001101110	4240	37500
83	0011110111	4620	32000
86	1001101110	4240	37500
88	0111110111	4620	32000
94	0111111111	4230	35600
95	1101110111	4620	32000
98	1101110111	4620	32000

Case 5.

Table 7. Results of the computer program, where the grey rows are Pareto-optimal front

ID	Bit string	Time to failure [h]	Cost [USD]
0	0101101010	3830	35000
1	0011110111	4620	32000
3	0101101011	4170	35900
5	1011010111	4650	31100
6	1011010111	4650	31100
7	0101101010	3830	35000
12	1010110111	4690	28600
13	1111011001	4180	34700
14	1011100010	4660	33000
16	1001111111	4230	35600
17	0101110111	4620	32000
18	0011110111	4620	32000
19	1101110111	4620	32000
20	0101101011	4170	35900
22	0101101011	4170	35900
23	1011010111	4650	31100
27	1111011111	4230	35600
28	0011110111	4620	32000
29	1101110111	4620	32000
30	1001111011	4230	35600
31	1011010101	5010	32700
32	0101101011	4170	35900
33	1001110111	4620	32000
34	1011010101	5010	32700
35	1101110111	4620	32000
36	1011101010	4380	34100
37	1011100010	4660	33000
38	0101110011	4620	32000
39	1111011111	4230	35600
40	0001111011	4230	35600
41	0111100011	5120	37000
42	1011010111	4650	31100
43	1011010101	5010	32700
45	0011111111	4230	35600
46	1001111011	4230	35600
47	1111011111	4230	35600
48	1011010101	5010	32700
49	1010110111	4690	28600
51	0101101010	3830	35000
52	1011101010	4380	34100
53	1101110111	4620	32000
54	0101101010	3830	35000
55	0111101011	4170	35900
56	0111101011	4170	35900
57	0001110111	4620	32000
61	0101101010	3830	35000
62	1101110111	4620	32000
63	0011110111	4620	32000
65	1011010111	4650	31100
67	0011111111	4230	35600

According to the results given in *Tables 3-7* and shown graphically in *Figures 8-12* the exemplary structures of the two-state parallel-series are presented in *Figures 13-17*.

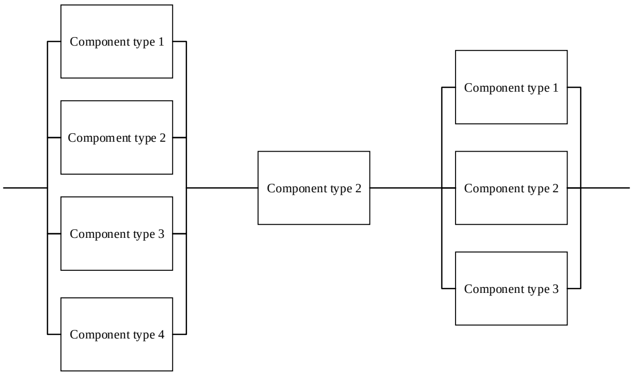

Figure 13. Example of the system structure according to results of the optimization in Case 1

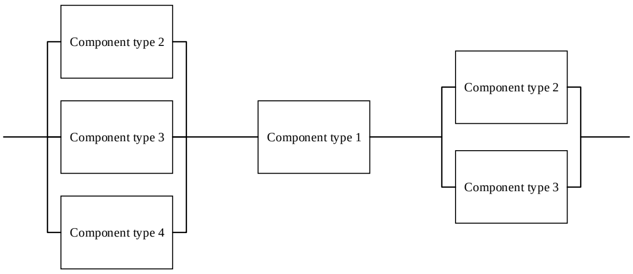

Figure 14. Example of the system structure according to results of the optimization in Case 2

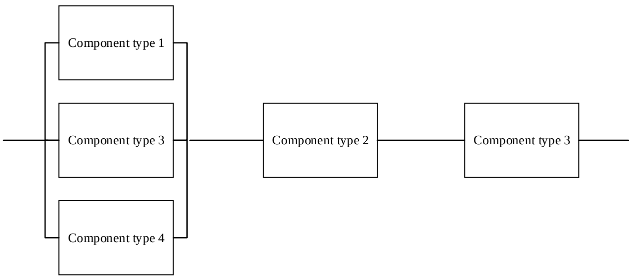

Figure 15. Example of the resulting system structure according to results of optimization in Case 3

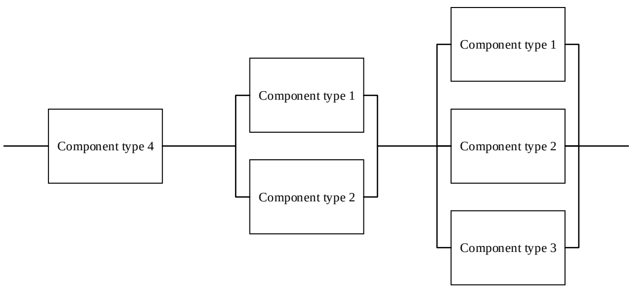

Figure 16. Example of the system structure according to results of the optimization in Case 4

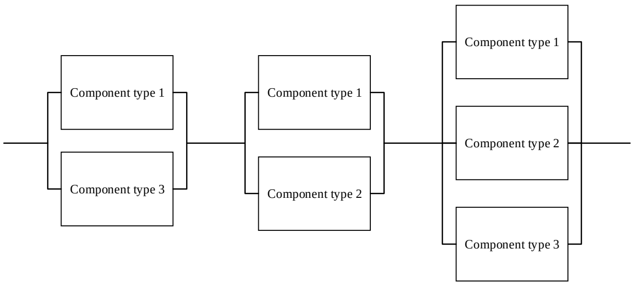

Figure 17. Example of the system structure according to results of the optimization in Case 5

These selected figures indicate a variety of opportunities to redesign the considered system with accordance to the time to failure and cost. Numerical data for these two objectives is given in *Tables 3-7*.

6. Conclusions

The SPEA algorithm and the binary knapsack problem have been described. The computer program to solve this problem based on this algorithm has been presented. Furthermore, the conversion of the reliability optimization problem to the 0-1 knapsack optimization problem has been proposed. Finally, the application of the computer program to the multi-criteria optimization for reliability problem has been done. The methods, algorithms and computer program presented in the paper can be applied to the reliability and safety optimization. The example in Section 5 has only shown potential applications of proposed computer program. In the future the extension of the capabilities of computer program for multi-objective optimization of the multi-state systems reliability should be done.

References

[1] Dziula, P., Kolowrocki, K., Siergiejczyk, M. (2014). *Critical infrastructure systems modeling*, Journal of Polish Safety and Reliability Association, Summer Safety and Reliability Seminars, 5, 1, 41-45, 2014.

[2] Guze, S. (2014). *Application of the knapsack problem to reliability multi-criteria optimization.* Journal of Polish Safety and Reliability Association, Summer Safety and Reliability Seminars, 5, 1, 85–90, Gdańsk-Sopot.

[3] Kołowrocki, K. (2004). *Reliability of Large Systems*. Elsevier, Amsterdam - Boston - Heidelberg - London - New York - Oxford - Paris - San Diego - San Francisco - Singapore - Sydney - Tokyo.

[4] Kołowrocki, K. & Soszyńska, J. (2010). *Optimization of complex technical systems*

operation processes. Maintenance Problems, 1, 31-40, Radom.

[5] Kołowrocki, K. & Soszyńska-Budny, J. (2011). *Reliability and Safety of Complex Technical Systems and Processes, Modeling – Identification – Prediction – Optimization,* Springer-Verlag.

[6] Kołowrocki, K. & Soszyńska-Budny, J. (2013). *Reliability prediction and optimization of complex technical systems with application in port transport*. Journal of Polish Safety and Reliability Association, Summer Safety and Reliability Seminars, 3, 1-2, 263 – 279, Gdańsk-Sopot.

[7] Kołowrocki, K. & Soszyńska-Budny, J. (2014). *Operation and reliability optimization of complex technical systems.* Journal of Polish Safety and Reliability Association, Summer Safety and Reliability Seminars, 5, 1, 91–106, Gdańsk-Sopot.

[8] Kuo, W. & Prasad, V.R. (2000). *An annotated overview of system-reliability optimization.* IEEE Trans. on Reliability 49, 2, 176-187.

[9] Kuo, W. & Zuo, M. J. (2003). *Optimal Reliability Modeling: Principles and Applications.* Hoboken: John Wiley & Sons, Inc.

[10] Lisnianski, A. & Levitin, G. (2003). *Multi-State System Reliability. Assessment, Optimisation and Applications.* World Scientific Publishing Co. Pte. Ltd.

[11] Martello, S. & Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. Chichester, U.K.: Wiley.

[12] Ming-Hua, L., Jung-Fa, T. and Chian-Son Y. (2012), *A Review of Deterministic Optimization Methods in Engineering and Management*, Mathematical Problems in Engineering, Volume 2012.

[13] Siergiejczyk, M. & Dziula, P. (2013). *Selected aspects of acts of law concerning crisis management and critical infrastructure protection.* Journal of Konbin, 2 (26), p. 79-88.

[14] Sun, W. & Yuan, Y.X. (2006). *Optimization theory and methods: nonlinear programming*. Springer-Verlag.

[15] Szłapczyńska, J. (2013). *Multicriteria Evolutionary Weather Routing Algorithm in Practice*. TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation, Vol. 7, No. 1, 61-65, Gdynia.

[16] Venter, G. (2010). *Review of Optimization Techniques, Encyclopedia of Aerospace Engineering,* John Willey and Sons, Ltd.

[17] Zitzler, E. (1999). *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications***.** PhD thesis, ETH Zurich, Switzerland.

[18] Zitzler, E. & Thiele, L. (1999). *Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach*. IEEE Transactions on Evolutionary Computation, VOL. 3, 4, 257–271.

[19] Website of PISA project available at www.tik.ee.ethz.ch/sop/pisa/?page=pisa.php.