# Optimisation of cyber insurance coverage with selection of cost effective security controls.

Ganbayar Uuganbayar[a,b], Artsiom Yautsiukhin[a], Fabio Martinelli[a], Fabio Massacci[b,c]

[a] *Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche, Pisa, Italy*
[b] *Department of Information Engineering and Computer Science (DISI), University of Trento, Italy*
[c] *Department of Computer Science, Vrije Universiteit, Netherlands*

**Abstract**

Nowadays, cyber threats are considered among the most dangerous risks by top management of enterprises. One way to deal with these risks is to insure them, but cyber insurance is still quite expensive. The insurance fee can be reduced if organisations improve their cyber security protection, i.e., reducing the insured risk. In other words, organisations need an investment strategy to decide the optimal amount of investments into cyber insurance and self-protection.

In this work, we propose an approach to help a risk-averse organisation to distribute its cyber security investments in a cost-efficient way. What makes our approach unique is that next to defining the amount of investments in cyber insurance and self-protection, our proposal also explicitly defines how these investments should be spent by selecting the most cost-efficient security controls. Moreover, we provide an exact algorithm for the control selection problem considering several threats at the same time and compare this algorithm with other approximate algorithmic solutions.

*Keywords:* cyber insurance, security investment, risk management, risk treatment, dynamic programming, genetic algorithm

## 1. Introduction

Cyber security losses due to successful cyber attacks grow every year [1, 2]. This increase can be explained by a number of factors, such as increasing reliance of business and society on IT systems, fast growth of the number of interconnected end devices, lack of security awareness, maturation of the cyber

*Email addresses:* `ganbayar.uuganbayar@{iit.cnr.it, unitn.it}` (Ganbayar Uuganbayar), `artsiom.yautsiukhin@iit.cnr.it` (Artsiom Yautsiukhin), `fabio.martinelli@iit.cnr.it` (Fabio Martinelli), `fabio.massacci@ieee.org` (Fabio Massacci)

crime world [3, 4], etc. These losses have already spurred top management of companies to consider cyber security risks among the most significant ones.

Typically, organisations have 4 basic options to treat risks: *avoid* it by abandoning risky business, *reduce* it by implementing security controls, *transfer* it to another entity (e.g., buy insurance), *accept* it if there is no other suitable choice [5, 6, 7]. The decision about treating cyber risks must be taken after careful analysis of the available options and ensuring that the selected strategy is cost-effective. Among the four risk treatment options, risk reduction and risk transferring require particular attention, because both of them require additional investments. Thus, organisations need an instrument which helps them to identify the required investments in different risk treatment strategies to minimise the expected losses.

Cyber insurance has received a lot of attention recently as more and more insurance companies enter the cyber market and their products become more mature. Also, numerous research papers devote significant attention to this topic [8, 9, 10, 11]. For instance, [8, 11] systematically reviewed the problems related to cyber insurance, where [11] points out that the area needs more technical-oriented solutions to replace qualitative analysis. Some researchers further investigated the relation between cyber insurance and security investment. For instance, Massacci et al., [12] highlighted that cyber insurance might trigger the drop of security investment when attackers are fully informed about the security posture of an organisation. On the other hand, several works are devoted to making cyber insurance market profitable and encouraging organisation to invest in self-protection [33, 34, 8]. The models used in the literature help to identify the most appropriate amount of investments in insurance and self-protection. These investment models simply assume that the probability of an attack is dependent on the amount of investment. In other words, the models do not tell how the organisation should spend these investments, i.e., how to select the required security configuration. Moreover, these models consider one threat (and one investment to probability dependency) while in reality organisation faces several cyber threats of a different kind which may cause different losses (e.g., see examples of threats from the ISO27005 standard [18]).

Risk reduction requires definition and implementation of a security configuration by the installation of various security controls to reduce the risk. Thus, an optimisation problem should be solved here, to select the most cost-efficient controls. For instance, in [7, 16, 17], the authors defined potential threats and controls in different systems, while an organisation may find it difficult to implement all required controls.

A typical mathematical problem to address the variety of choices is the family of knapsack problems [19, 20, 21, 22]. A direct formulation for selection of cyber security configuration [23] has several limitations. First, the knapsack problem will always use as much of the budget as the limit allows (taking into account the discrete costs for security controls), even if the application of security controls provide fewer benefits (in the reduction of cyber risks) than their costs. Naturally, this approach is not the most cost-effective.

Second, usage of risk as the utility function makes the solution computationally ineffective (i.e., existing pseudo-polynomial solutions, like dynamic programming, cannot be applied). In some specific cases, if just one threat is considered ([1, 24] efficient solutions are possible, but for a comprehensive case the computation is not trivial. Therefore, many researchers apply approximate solutions (e.g., Genetic Algorithms (GA) [25, 26]), accepting the risk of receiving the only near-optimal answer. We look for a satisfying solution in the sense of Simon [27].

### *1.1. Main contributions*

The core contribution of this paper is an approach for risk-averse organisations to determine security investments in various risk treatment options (in particular for risk reduction and cyber insurance) and determine specific security controls to be applied. In particular, this article provides

- a theoretical analysis of the distribution of the cost-efficient investments if cyber insurance is available,
- an explicit model to link investments with selected security controls, assuming that some initially selected controls could be not optimal,
- an exact solution for solving the optimisation problem (based on multi-objective knapsack problem),
- analysis of the proposed algorithms and approximate solutions (Greedy and GA), considering the effect of quantity and quality of inputs on the results.

The theoretical analysis is based on the utility theory and assumes a competitive insurance market in place. The resulting optimisation problem is represented as a multi-objective knapsack problem where minimisation of risks for every threat is a separate objective. The algorithmic solutions for the problem are based on the dynamic programming [23]. Moreover, we enhance the algorithm by applying our projection idea and compare the results by conducting several experiments.

This paper is organised as follows. We start with the analysis of the related literature in Section 2. Section 3 is dedicated to the formal problem statement definition and detailed discussion on its applicability. Section 4 is devoted to the adaptation of the existing algorithmic solutions for our problem. Section 5 provides experimental results comparing the proposed algorithms with some approximate solutions (i.e., Greedy and Genetic Algorithms) and analyse their applicability. Section 6 systematically discusses the assumptions we consider in the paper and proposes further directions to extend our work. Finally, conclusion and achieved results are outlined in Section 7.

## 2. Related work

To properly manage cyber risks, organisations should assess their risks [28, 29] and define an efficient risk investment strategy for treating them [7, 16]. In this work, we assume an organisation to be risk-averse and consider the option of *insuring* some of its risks. To minimise the insurance premium, the organisation should invest in self-protection and *optimise these security investments* by selecting the most cost-efficient security controls. In contrast to many existing papers, not only do we (prove and) formulate the problem for optimisation, but also propose our *algorithmic solution* which is adapted for our concrete problem (rather than being generic) and returns the optimal (exact) answer.

### *2.1. Cyber insurance*

In most surveys and annual reports [9, 30], cyber insurance market is gradually growing and positively impacting on cyber security, although it faces some challenges [8, 31]. Also, Savino Dambra et al., [11] underlined that current approaches for risk assessment are mostly qualitative and do not provide monetary information required for the cyber insurance underwriting process. The authors argue that the situation can be improved by applying data-driven methodologies and development of automatic tools. Scientific studies are largely focused on the problem of incentivising organisations to increase their cyber security investments if cyber insurance option is available [13, 14, 15, 32, 33]. Some researchers [32, 35, 12] claim that an organisation may invest less once the cyber insurance option becomes available. On the other hand, others [33, 34], i.e., Ruperto P.Majuca et al., found out that organisations are encouraged to invest for the self-protection to obtain a lower premium. Mukhopadhyay et al., [36] highlights that basic and concrete security investments for security controls are mapped to the affordable insurance premium. In contrast to the existing work on cyber insurance, we provide a cost-efficient approach for selecting concrete security controls and define how to map it with investment-probability correlation. To best of our knowledge, this problem has been often avoided by researchers and has not been investigated thoroughly.

### *2.2. Optimisation of security configuration*

The problem of selecting the best security controls received a lot of attention from various researchers. For example, Chung et al., [37] considered selecting the optimal security controls based on their Return on Investment (RoI) metric. This approach is similar to our Greedy algorithm and, as our experiments show, it often leads to only near-optimal answers.

Several authors [21, 38, 39, 40] focused on formulating the problem of optimising best controls selection where they applied the basic knapsack problem [19]. Even though the main concept is similar, the authors assumed various limitations and modifications of the knapsack problem. In particular, T. Sawik [38] considered minimization of expected worst-case cost using Value at Risk (VaR) and Conditional Value-at-Risk (CVaR) values. Lee at al., [41, 42] considered the return on investment, expected losses and required resources as additional constraints. Dewri et al., [39] and Viduto et al., [43, 44] formulated a problem considering both residual risk and cost of security controls as separate objectives to be minimised. Smeraldi and Malacaria [21] considered different assets to protect and binary interdependency of control. Fielder et al., [20] and Dewry et al., [39] merged a knapsack representation of the problem with a game-theoretic approach to decide which strategy is better for protecting against an attacker. Rees et al., [25] considered uncertainty in the optimisation problem and used fuzzy values for risk computation.

Many of these formulated problems are considered as multi-objective problems [21, 38, 39, 43]. We should underline that objectives considered by the authors (minimisation of losses, minimisation of costs, maximisation of return of investments, etc.) differ from our vision of objectives (i.e., minimising the risks caused by each threat). In this article, we prove that to optimise investments in self-protection and insurance we must consider total expenditure (i.e., minimise the sum of insured risk and total cost of controls).

The core difference of our paper from these works is that we propose an exact algorithmic solution based on dynamic programming and adapt it for our problem, while the authors of the mentioned papers simply apply existing off the shelves solutions. For example, [38] applied mixed-integer programming approach to analyse a simple example with 10 threats and 10 controls (something, our algorithm can easily cope with). Smeraldi [21] considered classic dynamic programming and greedy algorithms. Other authors [25, 39, 40, 44] used various evolutionary algorithms. Our experiments prove that GA is faster and reliable enough (with high settings), although it still may fail to produce the optimal answer. Viduto et al., [43] provided their solution called Multi-objective Tabu Search (MOTS), but their solution solves a slightly different problem which considers residual risk and security controls objectives separately.

### *2.3. Algorithmic solutions for optimisation problems*

Dynamic programming [45, 46] is one of the main solutions for the knapsack problem as it finds the optimal answer through iterations until a certain condition meets. In the work of Bazgan et al., [23], 0-1 multi-objective knapsack problem (considered 3 objectives for the experiment) has been solved by proposing dynamic programming, where the authors used several complementary dominance relations at different states and conducted experimental validation. In our paper, we improved the algorithm taking into account the peculiarities of our concrete problem (i.e., projection idea). On the other hand, evolutionary algorithms, i.e., Genetic Algorithm, are often considered in both literature and applications due to its capability of finding the optimal or near-optimal solutions in a much shorter time [47, 48].

There is several attempts to improve the performance of GA by many researchers [22, 26, 44, 49, 50]. Maya Hristakeva and Dipti Shrestha [22] proposed a GA-based solution for 0-1 knapsack problem and compared two selection methods, roulette-wheel and group selection (they devised a name for their approach). As an outcome, they claim that the group selection with elitism (copying some chromosomes without doing crossover technique) outperforms the roulette-wheel selection in different cases, i.e., increasing the number of population. Another work to improve the accuracy of GA, Gupta et al., [26] proposed a hybrid solution to create a better first population. They applied "fcheck" function to

| | | | |
|---|---|---|---|
| $W^0$ | - initial wealth | $\vec{\pi}$ | - probability of a threat *survival* |
| $x$ | - security investment | $\vec{F}$ | - expected number of threats *attempts* |
| $c$ | - cost of a control | $\vec{p}$ | - probability of a threat *occurrence* |
| $P$ | - premium | $\vec{z}$ | - *number* of threat occurrences |
| $K$ | - a set of *available* controls | $\vec{L}$ | - loss |
| $K_s$ | - a set of *selected* controls | $\vec{I}$ | - indemnity |

Table 1: Notations adopted in this work

the initialization step to check whether the created population meets the criteria they set. The idea results in more good chromosomes in the population, which eventually improved the accuracy of the outcome. Also, Ahmad et al., [49] proposed a linear regression analysis for creating the most efficient and fit population, yet this work is dedicated to another problem, travelling salesman problem (TSP). We have adapted some ideas of the aforementioned works to initialize the first population and improved them to fit into our work.

Also, GA has been applied to various applications, including cyber security. For instance, Suhail Owais et al., [51] surveyed to apply GA to Intrusion Detection Systems (IDS) techniques and Goranin et al., [50] adapted GA to find the controls to mitigate the propagation of worms through the Internet.

## 3. Problem specification

In this section, we formalise our main problem gradually adding the required features. First, the optimisation problem for the security of a system is described, then the insurance option is added and the whole model is considered from the point of view of utility theory in the usual way for analysis in the insurance literature [8, 32, 35]. The main parameters used in this paper are listed in Table 1.

### *3.1. No insurance case*

Consider an organisation which has conducted risk identification phase of risk assessment and has identified $n_t$ ($n_t \in \mathbb{N}^+$) relevant threats. For each threat, a corresponding expected loss has been defined $\vec{L} = \langle L^1, L^2, ..., L^{n_t} \rangle$, where $\vec{L}$ is a vector and $L^i (1 \leq i \leq n_t)$ is its i-th member. To decrease the amount of losses, the organisation invests $x$ in self-protection. This investment will be spent for installation and application of a set of security controls $K_s$, which can be seen as a subset of the set of all possible security controls $K$ (e.g., the ones that could be found in ISO27002 [52] or NIST 800-53 [7]). Let the cost of a control be a function and its result be a finite non-negative value $c : K \mapsto \mathbb{N}^+$ (i.e. thousands of Euro). The overall cost of installed controls $K_s \subseteq K$ ($c(K_s)$) is computed as:

$$c(K_s) = \sum_{\forall k \in K_s} c(k). \tag{1}$$

The probability of threat $i$ successfully passing through all installed security controls is denoted as $p^i(K_s)$, which eventually can be seen as a vector for all $n_t$ threats, $\vec{p}(K_s) = \langle p^1(K_s),\ p^2(K_s), ...,\ p^{n_t}(K_s)\rangle$. Now, if we know the frequency of threats occurrences $\vec{F} = \langle F^1, F^2, ..., F^{n_t}\rangle$, the overall risk for the organisation can be found as follows.

$$Risk(K_s, x, \vec{L}) = (\vec{F} \odot \vec{p}(K_s)) \times \vec{L}, \tag{2}$$

where $\vec{a} \times \vec{b}$ is a usual matrix multiplication of two vectors given as $\vec{a} \times \vec{b} = \sum_{i=1}^{n_t} a^i \cdot b^i$ and the Hadamard product of two vectors $\vec{a}$ and $\vec{b}$ is a vector $\vec{c} = \vec{a} \odot \vec{b} = \langle a^1 \cdot b^1, a^2 \cdot b^2, ..., a^{n_t} \cdot b^{n_t}\rangle$. In this paper, we omit the symbol for the transposition of vectors, required for proper representation of matrix multiplication, to simplify the formalisation since this precision is not crucial for the understanding of the paper.

Since a threat may occur more times than once in an observed period, and harm the organisation more than once, we need to take into account the distribution of probabilities with respect to the number of threat occurrences. Let $\vec{z} = \langle z^1, z^2, ..., z^{n_t}\rangle$ be a random vector of numbers of threat occurrences (one per threat) and $p(\vec{z}|K_s)$ be the probability that the considered organisation will face $\vec{z}$ incidents for some period of time conditional on the implemented controls $K_s$. If the organisation has $W^0$ as an initial wealth (or expected benefit from its core business), the wealth after occurrence of $\vec{z}$ threats can be defined as:

$$W(\vec{z}, K_s, x) = W^0 - x - \vec{z} \times \vec{L} \tag{3}$$

The goal of the organisation is to maximise its expected wealth, i.e.,

$$E[W(\vec{z}, K_s, x)] = \sum_{\forall \vec{z}} (W^0 - x - \vec{z} \times \vec{L}) \cdot p(\vec{z}|K_s) =$$
$$W^0 - x - \sum_{\forall \vec{z}} (\vec{z} \cdot p(\vec{z}|K_s) \times \vec{L} \tag{4}$$

We note that $\sum_{\forall z}(\vec{z} \cdot p(\vec{z}|K_s)^1$ is the mean number of occurrences, previously defined as $\vec{F} \odot \vec{p}(K_s)$. Finally, our optimisation problem can be seen as

$$\max_{x,K_s} E[W(\vec{z}, K_s, x)] = W^0 - x - (\vec{F} \odot \vec{p}(K_s)) \times \vec{L} = W^0 - x - Risk(K_s, x, \vec{L}) \tag{5}$$

*or*

$$\min_{x,K_s} [x + (\vec{F} \odot \vec{p}(K_s)) \times \vec{L}] \tag{6}$$

*3.2. Insurance case*

If an organisation buys insurance, it pays some premium $P$ regularly and expects some coverage of a loss if a threat occurs (called indemnity $\vec{I}$). Indemnity

[1]"$\cdot$" is the scalar multiplication defined as $\vec{a} \cdot b = \langle a^1 \cdot b, a^2 \cdot b, ..., a^{n_t} \cdot b\rangle$

is also a vector of size $n_t$, since depending on the purchased insurance product, different threats may get different coverage. The coverage is always lower than the loss itself, i.e., $\forall i,\ I_i \leq L_i$. In cyber insurance, a premium is usually computed through the estimated risk. A usual assumption is to consider a competitive insurance market [8] for which the premium is equal to risk for the insurer (i.e., $P = Risk(K_s, x, \vec{I})$). In case of purchased insurance, the resulting wealth for the organisation after occurrence of $\vec{z}$ threats (i.e., similar to Equation 4) can be found as:

$$W(\vec{z}, K_s, x, \vec{I}) = W^0 - (\vec{F} \odot \vec{p}(K_s)) \times \vec{I} - x - \vec{z} \times (\vec{L} - \vec{I}), \tag{7}$$
$$where\ \vec{I} - \vec{L} = \langle I^1 - L^1, I^2 - L^2, ..., I^{n_t} - L^{n_t} \rangle.$$

Similar to other economic models [8, 32, 35], we assume an organisation to be risk-averse and use utility of possessing a certain amount of wealth $U(W)$ instead of the wealth $W$ itself. The utility function is considered to be continuous, non-decreasing, and concave, i.e., $U'(W) > 0$ and $U''(W) < 0$.

$$U(W(\vec{z}, K_s, x, \vec{I})) = U(W^0 - (\vec{F} \odot \vec{p}(K_s)) \times \vec{I} - x - \vec{z} \times (\vec{L} - \vec{I})). \tag{8}$$

Finally, the expected utility is equal to:

$$E[U] = \sum_{\forall \vec{z}} p(\vec{z}|K_s) \cdot U(W^0 - (\vec{F} \odot \vec{p}(K_s)) \times \vec{I} - x + \vec{z} \times (\vec{I} - \vec{L})). \tag{9}$$

Our goal transforms into maximisation of the expected utility ($E[U]$) by selecting the optimal $x$, $\vec{I}$ and $K_s$.

In these settings, it is possible to prove that the optimal $\vec{I}$ for an organisation is equal to $\vec{L}$ (see the proof in the Appendix A). In other words, a risk-averse organisation insures all risks left after reduction if insurance option is available. This allows us to reduce Equation 9 to:

$$\max_{x, K_s} U(W^0 - x - (\vec{F} \odot \vec{p}(K_s)) \times \vec{L}). \tag{10}$$

Since the utility function is non-decreasing, we need to maximise its argument, or simply minimise the following part, which we call *expenditure* in the following:

$$\min_{x, K_s} (x + (\vec{F} \odot \vec{p}(K_s)) \times \vec{L}). \tag{11}$$

At this point we need to note, that our problem becomes equivalent to the risk-neutral approach, i.e., if $U(W) = W$ as in Section 3.1, for selecting security budget distribution and controls selection if no insurance is available (in this case, the premium becomes the accepted residual risk). The organisation should simply minimise its investments in self-protection and residual risk. Thus, our further contribution could be applied if one of the conditions (Equations 6 and 11) described above is found to be applicable.

### *3.3. Selection of security controls*

Let $K_s|x$ denotes a set $K_s$ which minimises Equation 11 for some fixed investment $x$. To minimise Equation 11 (as well as the minimisation condition in Equation 6), we need to compute $p(K_s|x)$ and determine the procedure for selection of $K_s$ in a way to minimise this component and ensure that we do this with investments less or equal to $x$.

Let $\pi^j(k) \in [0;1]$ be the probability that a threat $j$ passes through (survives) control $k \in K_s$; control $k$ completely eliminates threat $j$ if $\pi^j(k) = 0$, and is entirely powerless against the threat if $\pi^j(k) = 1$. Let $\vec{\pi}(k)$ be a vector of all probabilities of survival if control $k$ is installed, and the overall probability of survival $\vec{p}(K_s)$ can be computed as[2]:

$$\vec{p}(K_s) = \prod_{\forall k \in K_s} \vec{\pi}(k), \tag{12}$$

where $\prod_{\forall k \in K_s}$ stands for the Hadamard product.

Now, we say that $\vec{p}(K_s|x) = \vec{p}(K_s)$ if $K_s$ minimises Equation 11 and its overall cost is below $x$. Finally, the optimisation problem considered in this paper can be seen as:

$$\min_{\forall K_s \subset K} (\vec{F} \odot \left[ \prod_{\forall k \in K_s} \vec{\pi}(k) \right]) \times \vec{L} + x \quad and \quad \sum_{\forall k \in K_s} c(k) \leq x. \tag{13}$$

Equation 13 is the core problem we are tackling with in this paper.

*Our goal in this article is to find the efficient distribution of investments, i.e., self-investments, insurance and accepted risk. Furthermore, we go further and explicitly show how these investments should be used by selecting the best set of security controls to achieve this efficient budget distribution.* Considering the efficient distribution of investments and selection of controls helps us to achieve the desired cost-effectiveness, in contrast to efficient threat mitigation only considered in many of the related works [20, 21, 38, 39].

## 4. Algorithmic solutions

Equation 13 reminds of a 0-1 Knapsack problem, but uses multiplication (rather than summation) for aggregation of items (i.e., security controls) and has a complex utility function (multiplications and summations). This utility function is not order-preserving (i.e., if we add the same control to one set that was riskier than another one, the resulting set may become less risky). Moreover, we see that the budget limit is also a parameter of the utility function itself. This fact complicates the search for the solution.

In other words, although we see some obvious similarities between our problem and the 0-1 knapsack problem family, we need some modifications to the

[2]We assume effects of controls to be independent from each other.

available solutions to find the optimal budget distribution and select cost-efficient security controls.

First, we see our problem as the 0-1 multi-objective knapsack problem [23], i.e., a 0-1 knapsack problem with many utilities to maximise (i.e., threats to reduce, in our case). In our previous work [53], we adapted a dynamic programming solution into our problem. Although this solution was able to solve the problem, in theory, it is not very time- and resource-efficient. Therefore, in this paper we

- provide an improved version of the algorithm (e.g., embedding the projection idea),
- adapt a couple of approximate solutions (e.g., greedy and genetic algorithms) to our problem,
- conduct an analysis of the applicability of the three solutions.

## *4.1. Dynamic Programming*

We start with the dynamic programming for solving 0-1 multi-objective knapsack problem proposed by Bazgan et al., [23]. In general, dynamic programming could be applied if the main problem can be decomposed as recursively nested sub-problems.

First, we enumerate all the elements of $K$ as $j = 0, 1, ..., n_K$ (where $n_K$ is the size of $K$). We will sequentially try security controls, deciding whether to add the latest one into the selected set or reject it. In short, once we reach a control $k_q$, we will have all controls $j = 0, ..., q$ checked and continue with $j = q + 1, ..., n_k$.

Costs of controls can be represented as positive integer values, such that $\forall k \in K\ (c(j) = C \cdot m_j)$, where $C$ is the greatest common divisor for all costs and $m_j$ is just some positive natural value ($\forall j, m_j \in \mathbb{N}^+$). Similarly to controls, we will check the solutions for our problem gradually increasing the limit by $C$ (i.e., $x = C \cdot m$, where $m = 0, 1, ..., m_{max}$).

To advance in two directions (i.e. considering controls and budget limit) we need an auxiliary matrix $T$. Every cell $T[j][m]$ contains a solution of a sub-problem considering the first $j$ controls and budget limit $x = m \cdot C$ (the overall survival probability $\vec{p}(K_s|x),\ x = m \cdot C$ computed with Equation 12). Our goal is to consider all security controls and find the optimal budget $x^* = C \cdot m^*$ which leads to the minimal value of the first part in Equation 13. In other words, we are looking for the value (and associated selection of controls) of the cell $T[n_t][m^*]$. This will be our solution for the main problem, while every $T[j][m]$ (for $j < n_t$ and $m < m^*$) are the sub-problems.

Since the different combination of controls could fit into the budget limit, there could be several alternatives contributing to one cell in T. In the traditional method for solving 0-1 knapsack problem, every sub-problem (i.e., a selection of alternatives for $T[j][m]$ $j < n_t$ and $m < m^*$) could be solved, because the utility function used is order-preserving. Working with a multi-objective optimisation problem, we have no definitive criteria to select the best solution for a sub-problem.

Nevertheless, we may find some criteria which could help us to identify the solutions which are definitely worse than others, to minimise the number of alternatives. We call these solutions as *dominated*, and those solutions for which such a decision cannot be made are called *non-dominated*. Naturally, we should remove all dominated vectors to simplify the computation.

Basically, the core of our algorithm for 0-1 multi-objective knapsack problem could be seen as the following recursive algorithm:

1. $T[0][m] = 1$;
2. if $c(k_j) > m \cdot C$ (the new item is more expensive than the current cost limit);
   - Then:
     $T[j][m] = T[j-1][m]$
   - Else:
     $T_{add} = \bigcup_{\forall \vec{t} \in T[j-1][m-c(k_j)/C]} \vec{t} \odot \vec{\pi}(k_j)$
     $T[j][m] = non-dominated(T[j-1][m] \cup T_{add})$

Figure 1: Recursive algorithm

In our problem, we consider that security budget limit is not a given value, but is also a value to be optimised ($x^*$) by solving Equation 13.

It is worth noting that the recursive algorithm does not require the presence of the security investment bound. This fact allows us to start the investment from 0 and rise it until we find our solution (also extending matrix $T$ for new $x$ to check). In this regard, we should aim *to minimise the number of required iterations and ensure that the solution to Equation 11 will be found.*

We can re-write Equation 11 as follows, denoting the optimal premium (or residual risk) if $x$ amount is invested in self-protection as $P^*(x)$:

$$\min_{\forall x}(P^*(x) + x). \tag{14}$$

Consider some amount of investments $x_r \in [0, W^0]$ to be evaluated at step $r \in [0; W^0/C]$. We are interested only in the following future steps y:

$$x_r + P^*(x_r) > x_{r+y} + P^*(x_{r+y}); \tag{15}$$

$$x_{r+y} < P^*(x_r) + x_r - P^*_{min}, \tag{16}$$

$$where\ P^*_{min} = \vec{F} \odot \left[\prod_{\forall k \in K} \vec{\pi}(k)\right]) \cdot \vec{L}. \tag{17}$$

The aforementioned two equations (Equations 15 and 16) enable us to have the following observations. First, Equation 15, shows how to select the optimal value by comparing the current best value (i.e., up to step $r$) with the next ones ($y > 0$). The latter, Equation 16, defines the stopping point for the algorithm since there will be no more efficient solution if the condition fails. We also may find the first limit, which is: $x_0^{limit} = P^*(0) - P^*_{min}$, considering that $P^*_{min}$ is the minimal possible premium/risk that is computed with all possible controls $K_s = K$ installed. Naturally, if the company sets a limit for its investments $x^{lim}$ and $P^*(x_r) + x_r - P^*_{min} > x^{lim}$, we should bound our further steps with $x_{r+y} < x^{lim}$. Note that in this case we are not going to use all invested money, but look for a cost-effective solution within this budget.

We further note that the limit is reset for each better $x$, since it will be less than the previous one. This observation can be easily proved as follows. Let $x_r$ be the previous best value (i.e., for all $r + y - 1$ steps) and $x_{r+y}$ be even better than $x_r$, i.e.,:

$$P^*(x_r) + x_r > P^*(x_{r+y}) + x_{r+y}. \tag{18}$$

The limits defined at steps $r$ and step $r + y$ are $x_r^{limit}$ and $x_{r+y}^{limit}$ consequently:

$$P^*(x_r) + x_r - P^*_{min} = x_r^{limit} \ ; \quad P^*(x_{r+y}) + x_{r+y} - P^*_{min} = x_{r+y}^{limit}. \tag{19}$$

We conclude that $x_r^{limit} > x_{r+y}^{limit}$.

*Algorithm.* Now, we may write an algorithm, based on our ideas described above (Algorithm 1), which a) finds the optimal investments in self-protection $x^*$; b) ensures the lowest expenditure $((\vec{F} \odot \vec{p}(K_s^*|x^*)) \cdot \vec{L} + x^*)$. The algorithm is based on the dynamic programming approach to solve the 0-1 multi-objective knapsack problem [23]. Although the core part of the algorithm has been re-used, we have adapted it for our problem and make it return the optimal investment as an output, instead of taking it as an input.

At the initial phase, the algorithm requires all variables and functions for the input. One of the advantages of this algorithm is that it allows taking into account already invested funds $x_{init}$, spent for installation of some initial set of controls $K_{init}$ and reducing the probability of attack with $\vec{p}_{init}$. The initial controls are not used in our further analysis: $K_{init} \cap K = \emptyset$.

The algorithm starts (9 and 16) with setting up initial values and computing the minimal premium $P_{min}$ at this point. $GCD(\cup_{\forall k \in K} c(k))$ function returns the Greatest Common Divisor[3].

We gradually increase the limit $x$ (and its counter $m$) until the procedure reaches the limit $exp - P_{min}$, as Equation 16 states (line 17). For every limit $x$ we consider all controls one by one (line 19). For every control, we compare: 1) a set of previously selected controls with $k_j$ $(\bigcup_{\forall l} \vec{\pi}(k_j) \odot T[j-1][m - c(k_j)/C][l])$, 2) and the best selection of controls without $k_j$ $(T[j-1][m])$(line 21). We leave only non-dominated elements sets of controls.

[3]The algorithm for finding GCD is well known and is not included in the paper.

**Algorithm 1:** Selecting the best set of controls

```
1  Function(searchForOptimalInvesments)
   Input: $K$, $c$, $\pi$, $\vec{F}$, $\vec{L}$, $x_{init}$, $p_{init}$, C
   Require:
2  $K$                                                    // a set of controls
3  $c : K \mapsto \mathbb{N}$                             // cost function
4  $\pi : K \mapsto 2^{[0;1]}$                            // survival probability per threat function
5  $\vec{F}$                                              // frequency vector of $\mathbb{R}^+$ values
6  $\vec{L}$                                              // single loss expectency vector of $\mathbb{N}^+$ values
7  $x_{init} \in \mathbb{N}$                              // initial investments
8  $\vec{p}_{init}$                                       // initial probability of survival vector of values from $[0;1]$
   Ensure: $\min(\vec{F} \odot \vec{p}(K_s|x)) \times \vec{L} + x)$ for optimal security investment $x^*$
9  $exp := (\vec{F} \odot \vec{p}_{init}) \times \vec{L} + x_{init}$     // Initial expenditure as optimal
10 $P^*_{min} := \vec{F} \odot \left[\prod_{\forall k \in K} \vec{\pi}(k)\right]) \times \vec{L}$
11 $x^* := x_{init}$                                      // Optimal Investment starts with $x_{init}$
12 $\forall j\ T[j][0] :=\ \{\vec{p}_{init}\}$            // a dynamic matrix of optimal probabilities. Add
     new (and the first) column $x = x_{init}$, with one vector $\vec{p}_{init}$
13 $C := GCD(\cup_{\forall k \in K} c(k))$                // the greatest common divisor for costs
14 $x := C$                                               // first increase of investments
15 $m := 1$                                               // investment counter starts with 1
16 $n_k := |K|$                                           // the size of set $K$
17 while $x + x_{init} \leq exp - P^*_{min}$ do
                                                          // Do while $x$ is below the optimal expenditure
18     $\forall j\ T[j][m] :=\ \{\vec{p}_{init}\}$        // new column is set with vector $\vec{p}_{init}$
19     for $j := 1$ to $n_k$ do
                                                          // for all controls
20         if $(c(k_j) \leq x\ )$ then
                                                          // check the cost limit
21             $T[j][m] :=$
               $non-dominant \begin{cases} \bigcup_{\forall l} \vec{\pi}(k_j) \odot T[j-1][m - c(k_j)/C][l] \\ T[j-1][m] \end{cases}$
                   // store all non-dominant vectors comparing two sets: with
               new control and without.
22         else
23             $T[j][m] := T[j-1][m]$
                                                          // continue without adding new control $j$
24     for $l := 0$ to $|T[n_k][m]|$ do
                                                          // for all vectors stored in $T[n_k][m]$
25         if $(\vec{F} \odot T[n_k][m][l]) \times \vec{L} + x + x_{init} < exp$ then
                                                          // reduced the expenditure?
26             $exp := (\vec{F} \odot T[n_k][m][l]) \times \vec{L} + x + x_{init}$
                                                          // Store this expenditure as optimal
27             $x^* := x$                                 // Remember these investments as optimal
28     $x := x + C$
29     $m := m + 1$
   Return: $exp, x^*$
```

As we mentioned earlier, our solution is derived from classic dynamic programming yet, instead of summing the values, we multiply them and look forward to ensuring the lowest overreaching to the maximum. Furthermore, when we set an additional investment for the next column, the overall probability of survival should be computed considering the changes. In particular, if the cost of a control $k_j$ ($c(k_j)$) is higher than the additional investments $x$, we keep the previously selected controls as well as the corresponding survival probabilities $T[j-1][m]$ (line 23).

After considering all controls for current security investment $x$, we check whether the total expenditure computed by Equation 15 is lower than the previously computed one (line 25). If this condition is met by some vector from $T[n_k][m]$, we reset our optimal security investment to the new one (line 27) and, more importantly, in line 26, we further replace the expenditure with the new one for further computations (according to the condition in Equation 16).

We find the lowest possible expenditure and the optimal security investment at the last iteration before the condition in line 17 fails. To find the selected controls, a simple backward algorithm should be applied as we have in Algorithm 3 (see Appendix B).

*Dominance criteria.* The last part of our algorithm is the criteria for dominance. In our previous work [53], we used Pareto optimality, i.e., we checked that every element of one vector is better or equal (in our case, lower or equal) than the corresponding element from another vector. We studied further possible ways to improve our DP algorithm since the number of non-dominated vectors grows fast and this slows the algorithm, especially as the number of considered controls grows.

We have found two possible improvements, i) Projection and ii) Sorting. Projection aims to strengthen the dominance criteria, by looking to the remaining controls and assuming the worst case for a vector with lower risk. If the vector under these conditions still results in a lower risk than its opponent, we discard the second vector. Sorting security controls by their cost, leaving the costlier ones for later consideration, tends to reduce the number of possible alternatives (and non-dominated vectors) at the end.

To implement the projection idea, we first compute the best case values for all controls by multiplying values for threats starting from the latest control. We use a table $Best[][]$ to store these values, which are computed as:

$$Best[j][i] = \prod_{q=n_K-1}^{j} \pi(k_q)[i] \tag{20}$$

Next, for every set of already considered controls (e.g., by $j$-th one) we know the lowest survival probability if all remaining controls will be installed. Now, let us compare two vectors $\vec{p}$ and $\vec{p'}$ after considering only $j$-th control and see that risk for the first one is lower than for the second one, i.e.,

$$\vec{F} \odot (\vec{p} - \vec{p'})) \times \vec{L} < 0. \tag{21}$$

Let $\mathbb{I}$ be a set of all indexes for threats ($|\mathbb{I}| = n_t$) and for some $i \in \hat{\mathbb{I}} \subset \mathbb{I}$ $p[i] > p'[i]$ and for others $i \in \mathbb{I} \backslash \hat{\mathbb{I}}$ $p[i] \leq p'[i]$. We form a vector $\vec{D}$ with $n_t$ values, as follows:

$$D[i] = \begin{cases} Best[j][i] & if\ i \in \hat{\mathbb{I}} \\ 1 & if\ i \in \mathbb{I} \backslash \hat{\mathbb{I}} \end{cases} \tag{22}$$

We make the situation worse for the first vector by reducing the probabilities of survival for the threats from $\hat{\mathbb{I}}$ and check if the first vector still results in lower risk:

$$\vec{F} \odot (\vec{D} \odot (\vec{p} - \vec{p'})) \times \vec{L} < 0. \tag{23}$$

If that is the case, we can remove the second vector from the further consideration, since no matter what we apply in the future the risk computed using the first set of controls will always be lower. Note that this definition of dominance supersedes the former Pareto optimal approach. Algorithm 2 encodes this idea.

## *4.2. Greedy*

Our DP based solutions accurately find the optimal answer for our main problem, however, they require a lot of time to find the solution when there is a large number of input parameters. Alternatively, approximate solutions could be applied to find nearly optimal answers.

One of such approaches is the Greedy approach [54]. The idea behind the greedy approach is to add (or remove) the elements which separately contribute the most (or the least) to the overall goal. Such an approach does not guarantee to find the optimal value but is fast and easy to implement.

Our version of a Greedy algorithm (see Appendix B) starts with all controls to be selected and gradually remove the ones which removal reduces the overall expenditure more than the removal of others. We do this until we are not able to remove any control without increasing the expenditure.

## *4.3. Genetic Algorithm*

One of the most used optimisation approaches is the Genetic Algorithm (GA) [22, 25, 39, 40, 44], which provides the optimal or nearest optimal solutions in a short time. We basically keep the foundation of GA [22, 25], but make some changes to fit it into our problem (see the Algorithm 7 in Appendix B).

First, we randomly generate an initial population of chromosomes (every chromosome is an example of a security configuration with every gene representing a security control). This population is further used for generation of the new population which will be compared with the initial one with respect to the main criteria (i.e., Equation 11).

The most important part of GA is the creation of a new population (algorithm 9), where it comprises crossover and mutation procedures with a merge

**Algorithm 2:** Projection Function of the DP

```
1  Function(newDominanceCheck)          // Checking new dominance using
     projection idea
   Input: $VectorsA, VectorsB, Best[j], \vec{F}, \vec{L}, n_t$
   Require:
2  $VectorsA, VectorsB$        // two sets of vectors to check for dominance
3  $Best[j]$         // A vector with corrective values for the current step j
4  $\vec{F}$                                 // frequency vector of $\mathbb{R}^+$ values
5  $\vec{L}$                         // single loss expectency vector of $\mathbb{N}^+$ values
6  $n_t$                                                 // number of threats
   Ensure: A set of non-dominated vectors
7  for $q < |VectorsA|$ do
8      for $l < |VectorsB|$ do
9          $v1 := 0$                                   // vector q is dominating
10         $v2 := 0$                                   // vector l is dominating
11         for $k < n_t$ do
12             if $(VectorsA[q][i] - VectorsB[l][i]) < 0$ then
13                 $v2 := v2 + (VectorsA[q][i] - VectorsB[l][i]) \cdot F[k] \cdot L[i]$
14                 $v1 := v1 + (VectorsA[q][i] - VectorsB[l][i]) \cdot Best[j][i] \cdot F[k] \cdot L[i]$
15             else
16                 $v2 := v2 + (VectorsA[q][i] - VectorsB[l][i]) \cdot Best[j][i] \cdot F[k] \cdot L[i]$
17                 $v1 := v1 + (VectorsA[q][i] - VectorsB[l][i]) \cdot F[k] \cdot L[i]$
                                                   // new vector is dominating
18         if $v2 > 0$ then
19             remove vector q from $VectorsA$
                                      // old vector is dominating or the same
20         if $v1 <= 0$ then
21             remove vector l from $VectorsB$
22 result := $VectorsA \cup VectorsB$
```

function. We have improved the techniques for crossover by adapting both two-point and single-point methods as are shown in Figure 2. Then, the rest of the population is added to a new population list. Now, the new list of chromosomes will be used for crossover by using three different combinations – good with good, good with bad and bad with bad.

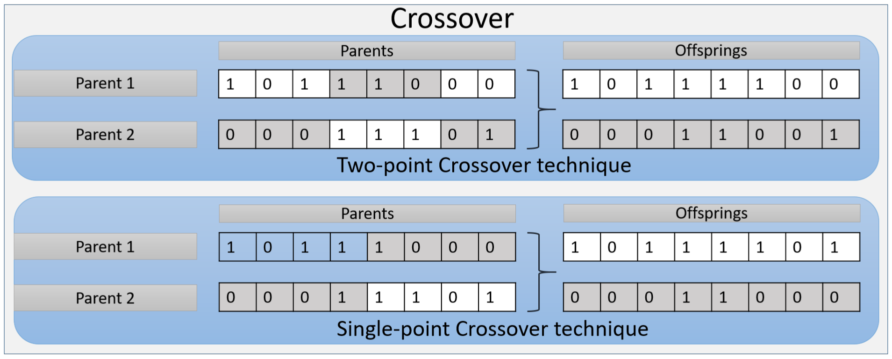

Figure 2: Crossover techniques

After that, the algorithm sorts the chromosomes and checks if the best chromosome found after the crossover procedure is greater than the one found before. In GA, mutation helps to overcome local minimum solutions and find global optima. We use the basic technique of mutation, selecting random bits (number of random bits is defined in prior) based on a defined number, with elitism technique, and reverse those bits as is shown in the Figure 3. This process should be done for the offsprings after crossover procedure, and sort them by their weights to find the best one.

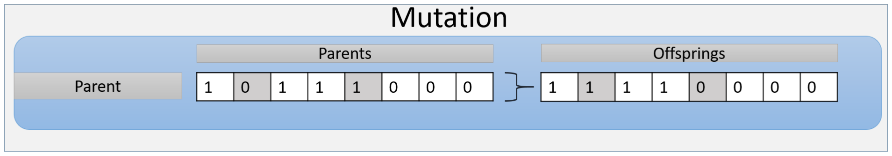

Figure 3: Mutation process

## 5. Experiments

In this section, we compare our solution with Greedy and GA in different scenarios.

The first experiment is to ensure whether the approximate algorithmic solutions and the improved DP yield the same result that we had in our previous work [53].

Furthermore, what we are interested in is the *execution time* of the solutions and its dependence on various input parameters. The second parameter which we would like to investigate is *accuracy* of the approximate solutions (Greedy and GA). We may expect the approximate solutions to perform better than our DP algorithms, but we would like to test the limits of the algorithms and verify that the version of the DP algorithms with projection idea is indeed beneficial.

To evaluate the accuracy of the approximate solutions, we need to know the exact answer, and therefore we compare the results of the approximate solutions with the result of the DP algorithm. If the computation becomes too slow for the DP-based algorithms to produce the result, we run the GA algorithm several times (e.g., high settings) and assume that the result which we receive most often is the correct one. The last assumption does not provide a 100 percent guarantee that the found result is the exact answer, but this is the best reasonable check we can do. Moreover, for this reason, we do not experiment with the values of the input parameters showing a high dispersion of results.

For the sake of testing, we have created a simple supporting program which randomly generates inputs for our problem. This generation is not fully random, but controlled, i.e., it depends on some quantitative and qualitative input parameters.

For more complicated cases, we start investigating how the quantity of the input parameters, i.e., the *number of considered controls* and *threats*, affects the execution time and accuracy of the considered solution. Then, we move to the analysis of the effect of quality of input, i.e., how the input is formed.

We consider the following qualitative input parameters. The *Greatest Common Divisor (GCD) C* determines the granularity of the search for a suitable solution (the higher the GCD is, the easier should be for the algorithms to compare the control costs). *Range of the control costs*, i.e., how different the costs are from each other, could make the solution look too much alike for the algorithms to find a global optimum. The *number of threats affected by a control* shows how many threats could be reduced by installing one control and thus, potentially, could have an impact on the projection idea.

All experiments have been run on a machine (Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz (8 CPUs), 2.0GHz) with Windows 10 operating system. We acknowledge that usage of another (e.g., more powerful) computing system will result in different speed of the proposed solutions, but we are more interested in studying the existing dependencies, than finding the exact time of execution for the algorithms.

## *5.1. Basic and Simple scenario*

We use a simple example to demonstrate the work of the proposed solutions. We consider an organisation which has identified five main threats and the corresponding single loss expectancy ($\vec{L} = \langle 3000, 1800, 2800, 4000, 3800 \rangle$). Eight available controls have been proposed ($|K| = n_k = 8$) to install with costs as ($c(k_1) = 480$; $c(k_2) = 240$; $c(k_3) = 120$; $c(k_4) = 80$; $c(k_5) = 200$; $c(k_6) = 120$; $c(k_7) = 280$; $c(k_8) = 200$). Finally, the initial probability of survival $\vec{p}_{init}$ (caused by already installed controls), expected frequency of threats $\vec{F}$ and the probabilities of survival for every proposed controls $\vec{\pi}_j$ are found as shown in Table 2.

As a result of applying all approaches, we obtain the best controls to keep the security expenditure at minimum. Our DP algorithm (see Figure 4) successfully passes the local minimums (i.e. for $x = 80$, $x = 320$ or $x = 560$) for expenditure and finds the global one (i.e., for $x = 760$). We refer the readers to see equations 18 and 19 for ensuring how the algorithm finds the global minimum and stops after some steps.

| $\vec{p}_{init}$ | $\vec{F}$ | $\vec{\pi}_{k1}$ | $\vec{\pi}_{k2}$ | $\vec{\pi}_{k3}$ | $\vec{\pi}_{k4}$ | $\vec{\pi}_{k5}$ | $\vec{\pi}_{k6}$ | $\vec{\pi}_{k7}$ | $\vec{\pi}_{k8}$ |
|---|---|---|---|---|---|---|---|---|---|
| 0.6 | 0.8 | 0.3 | 0.9 | 0.5 | 0.8 | 0.9 | 0.8 | 0.8 | 0.6 |
| 0.7 | 0.5 | 0.2 | 0.8 | 0.7 | 0.6 | 0.5 | 0.7 | 0.1 | 0.7 |
| 0.8 | 0.4 | 0.5 | 0.9 | 0.9 | 0.9 | 0.8 | 0.5 | 0.4 | 0.5 |
| 0.6 | 0.7 | 0.7 | 0.2 | 0.8 | 0.8 | 0.6 | 0.8 | 0.9 | 0.8 |
| 0.6 | 0.5 | 0.3 | 0.7 | 0.6 | 0.2 | 0.5 | 0.6 | 0.8 | 0.5 |

Table 2: Input vectors

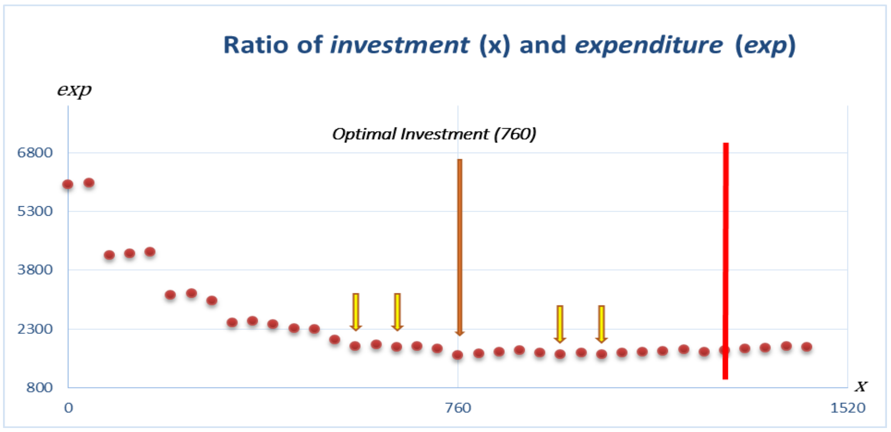

Figure 4: $(Exp)$ expenditure for security self-investments $x$

If an organisation selects $\{k_2, k_3, k_4, k_6, k_8\}$ and invests 760, the expenditure is 1642 which is the global minimum so far. Figure 4 also shows that the overall expenditure slowly increases after the optimal value (with some occasional small drops), which confirms that further investments in self-protection are not cost-efficient (even though they reduce the overall risk). Moreover, in order to avoid spending unnecessary resources for the computation, DP based algorithms stop running at $x = 1280$.

Both Greedy and GA algorithms find the same optimal answer in a shorter time than DP (as expected). We should underline that the inputs for this toy example are simple, but in the following experiments, we will consider larger and more complex examples in which these algorithms will have difficulty to find the right answer.

### *5.2. Quantitative input parameters*

To conduct more complicated experiments, we start with the analysis of the effect of the number of threats $n_t$. The inputs are generated randomly with 20 security controls, 40 GCD, the cost ranging from 80 up to 400, and each control affecting every threat. Since for GA various settings are possible, in order to limit the number of experiments and focus on the main outcomes, we set "high[4]" values for the GA settings (see Table 3) during this set of experiments. We will vary some of these settings investigating qualitative input parameters.

| | | | | | chromosomes combination | | |
|---|---|---|---|---|---|---|---|
| Mutation | Limit | Population size | Two point crossover percentage | Elitism percentage | Good & Good | Good & Bad | Bad & Bad |
| 1 bit | 1000 | 1000 | 80% | 15% | 50% | 45% | 5% |

Table 3: Constants for GA execution

Table 4 shows the results of our experiments.

| | | *Number of threats* | **5** | **10** | **15** | **20** | **25** | **30** | **35** | **40** | **50** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **20 controls and up to 50 threats** | **DP** | *Execution time (sec)* | 0.3s | 1.6s | 16.7s | 80.5s | 323s | 756.5s | | | |
| | | *Overall loss* | 800.2 | 825.4 | 963.6 | 1140.5 | 1213.8 | 1306.5 | | | |
| | **Projection** | *Execution time (sec)* | 0.1s | 0.2s | 0.7s | 6.8s | 8.9s | 18.1s | 27.4s | 58.5s | 271.9s |
| | | *Overall loss* | 800.2 | 825.4 | 963.6 | 1140.5 | 1213.8 | 1306.5 | 1479 | 1617.8 | 1638.9 |
| | **Greedy** | *Execution time (sec)* | 0.01s | 0.014s | 0.021s | 0.015s | 0.008s | 0.009s | 0.0156s | 0.016s | 0.015s |
| | | *Overall loss* | **914.3** | 825.4 | **1045.2** | 1140.5 | 1213.8 | **1317.1** | **1560.1** | 1617.8 | 1638.9 |
| | **GA** | *Execution time (sec)* | 2.8s | 3.7s | 5.1s | 6.9s | 8.5s | 9.7s | 10.9s | 14.1s | 16.4s |
| | | *Overall loss* | 800.2 | 825.4 | 963.6 | 1140.5 | 1213.8 | 1306.5 | 1479 | 1617.8 | 1638.9 |

Table 4: Both Time&Accuracy of the solutions for increasing number of threats

First, we analyse the dependency of the execution time on the number of considered threats $n_t$ (see Figure 5). We see that the Greedy algorithm is not affected very much by the number of threats, the time for execution for others grows with the increase of the number of threats. DP is affected the most and its time of execution rises quickly. Our improved DP algorithm with projection performs better, but still, its time of execution grows faster than the one for GA.

[4] "High Settings" for GA represents a larger number of population size and a round of iteration (limit)

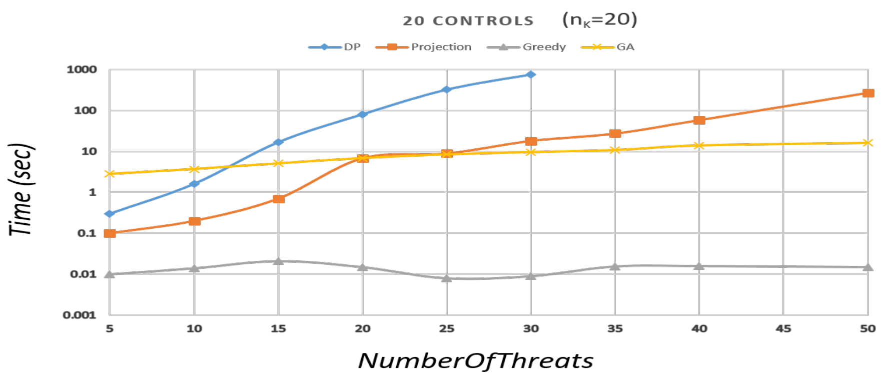

Figure 5: Comparison of execution time for 4 solutions in case of increasing number of threats with 20 control cases

Although the Greedy solution is the fastest among the four solutions and at the same time it fails more often (highlighted as ***bold*** in see Table 3) to find the optimal answer. GA with our settings performs well and always finds the optimal answer.

Next, we conduct 4 series of experiments, to analyse the dependency of the execution time on the number of controls. The number of threats considered is fixed in every experiment $n_t = 5, 10, 15$ and 20. The four graphs in Figure 6 represent the results of the time dependency while the Table 5 indicates both time and accuracy of the solutions.

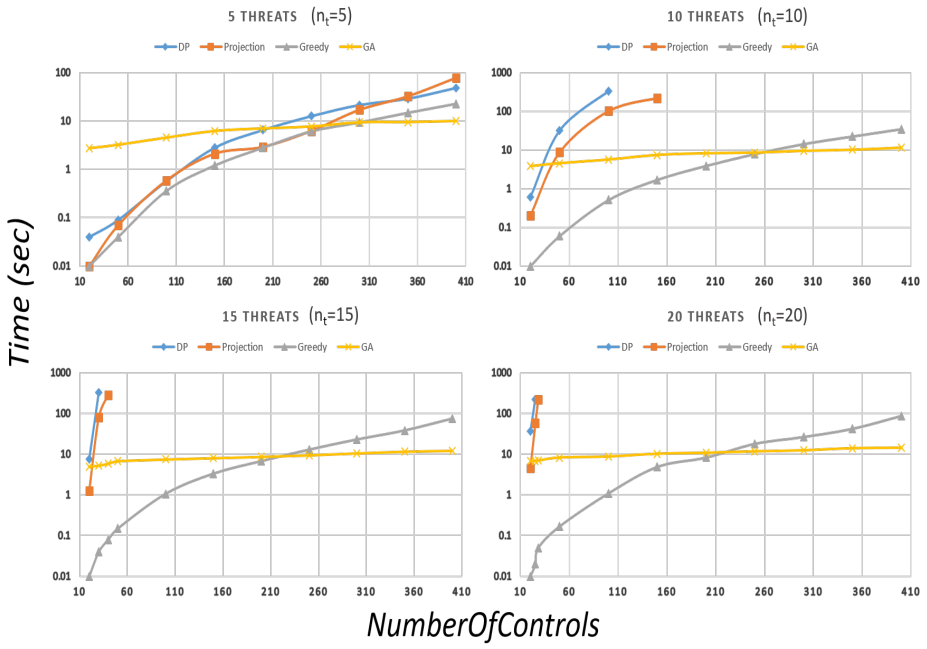

Figure 6: Comparison of execution time for 4 solutions in case of increasing number of controls with 5, 10, 15, 20 threats cases

The Greedy solution in all cases performs faster than others for a lower number of controls, but then its execution time increases. For instance, its execution time grows faster than GA, which outperforms the Greedy solution as the number of controls rises larger than about 200. Moreover, we see that GA is affected by the number of controls only slightly, and, thus, we may expect that its performance continues to stay the best and the execution time relatively fast.

DP algorithms, as was expected, are the slowest and are the most affected by the number of controls. Their execution time grows as $n_K$ grows especially when the number of considered threats is large. DP with projection outperforms the ordinary DP algorithm, but this difference is much lower than the difference with approximate algorithms.

Analysing accuracy, we can see that the Greedy solution often fails to yield the optimal answer whereas GA always finds the best ones. We admit that, at some points, it is complicated to say whether GA provides the optimal answer, however, we are confident because of the several times of running the GA.

Thus, we may conclude that the projection idea helps to improve the DP algorithm, but its effect is reducing with the increase of the input parameters. Also, Greedy and GA outperform the DP solutions and are able to provide the result in a reasonable time even for a high number of threats and controls. GA algorithm is less affected by the grows of parameters, although the Greedy one

|  |  | *Number of Controls* | **20** | **50** | **100** | **150** | **200** | **250** | **300** | **350** | **400** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **5 threats case with increasing number of controls** | *DP* | *Execution time (sec)* | 0.04s | 0.09s | 0.57s | 2.8s | 6.5s | 12.6s | 21.4s | 28.9s | 48.3s |
|  |  | *Overall loss* | 510.6 | 359.6 | 356.9 | 329.3 | 322.3 | 322.3 | 322.3 | 322.3 | 322.3 |
|  | *Projection* | *Execution time (sec)* | 0.01s | 0.07s | 0.59s | 2.1s | 2.9s | 6.2s | 16.9s | 32.1s | 77.4s |
|  |  | *Overall loss* | 510.6 | 359.6 | 356.9 | 329.3 | 322.3 | 322.3 | 322.3 | 322.3 | 322.3 |
|  | *Greedy* | *Execution time (sec)* | 0.01s | 0.04s | 0.36s | 1.2s | 2.8s | 6.1s | 9.3s | 14.7s | 22.6s |
|  |  | *Overall loss* | 510.6 | **413** | 356.9 | **353.7** | **337.9** | **337.9** | **337.9** | **342.8** | **363.7** |
|  | *GA* | *Execution time (sec)* | 2.7s | 3.2s | 4.5s | 6.2s | 7s | 7.69s | 9.4s | 9.5s | 10s |
|  |  | *Overall loss* | 510.6 | 359.6 | 356.9 | 329.3 | 322.3 | 322.3 | 322.3 | 322.3 | 322.3 |
| **10 threats case with increasing number of controls** | *DP* | *Execution time (sec)* | 0.6s | 31.9s | 331.4s |  |  |  |  |  |  |
|  |  | *Overall loss* | 850.1 | 553.7 | 468.1 |  |  |  |  |  |  |
|  | *Projection* | *Execution time (sec)* | 0.2s | 8.7s | 101s | 220.7s |  |  |  |  |  |
|  |  | *Overall loss* | 850.1 | 553.7 | 468.1 | 428.6 |  |  |  |  |  |
|  | *Greedy* | *Execution time (sec)* | 0.01s | 0.06s | 0.51s | 1.7s | 3.9s | 7.9s | 14.3s | 22.5s | 34.8s |
|  |  | *Overall loss* | 850.1 | **562.7** | **488.3** | 428.6 | 416.5 | 385.1 | **422.3** | **422.3** | **455.3** |
|  | *GA* | *Execution time (sec)* | 3.77s | 4.48s | 5.6s | 7.4s | 8.2s | 8.5s | 9.5s | 10.1s | 11.5s |
|  |  | *Overall loss* | 850.1 | 553.7 | 468.1 | 428.6 | 416.5 | 385.1 | 385.1 | 385.1 | 385.1 |
| **15 threats case with increasing number of controls** | *DP* | *Execution time (sec)* | 7.6s |  |  |  |  |  |  |  |  |
|  |  | *Overall loss* | 977 |  |  |  |  |  |  |  |  |
|  | *Projection* | *Execution time (sec)* | 1.24s |  |  |  |  |  |  |  |  |
|  |  | *Overall loss* | 977 |  |  |  |  |  |  |  |  |
|  | *Greedy* | *Execution time (sec)* | 0.01s | 0.15s | 1.06s | 3.31s | 6.7s | 12.8s | 22.7s | 37.8s | 74s |
|  |  | *Overall loss* | 977 | **834.2** | 598.8 | **599.2** | 466.4 | **478.4** | **488.5** | **471.9** | **441.9** |
|  | *GA* | *Execution time (sec)* | 4.9s | 6.7s | 7.4s | 8s | 8.6s | 9.4s | 10.5s | 11.6s | 12.2s |
|  |  | *Overall loss* | 977 | 789.7 | 598.8 | 576.2 | 466.4 | 466.4 | 468.5 | 465.2 | 434 |
| **20 threats case with increasing number of controls** | *DP* | *Execution time (sec)* | 36.7s |  |  |  |  |  |  |  |  |
|  |  | *Overall loss* | 1236.2 |  |  |  |  |  |  |  |  |
|  | *Projection* | *Execution time (sec)* | 4.6s |  |  |  |  |  |  |  |  |
|  |  | *Overall loss* | 1236.2 |  |  |  |  |  |  |  |  |
|  | *Greedy* | *Execution time (sec)* | 0.01s | 0.17s | 1.1s | 4.9s | 8.3s | 18.1s | 26.6s | 42.3s | 87.5s |
|  |  | *Overall loss* | 1236.2 | 850.9 | **774.2** | **699.2** | **678.6** | **645.3** | **653.8** | **667.7** | 577.9 |
|  | *GA* | *Execution time (sec)* | 6.7s | 8.2s | 8.7s | 10.2s | 10.8s | 11.7s | 12.4s | 14s | 14.3s |
|  |  | *Overall loss* | 1236.2 | 850.9 | 714.2 | 659.1 | 642 | 619.2 | 619.2 | 612.1 | 577.9 |

Table 5: Both Time&Accuracy comparison for increasing number of controls

is faster for smaller numbers. But, the Greedy algorithm usually provides an only nearly optimal answer.

## *5.3. Qualitative parameters*

We further analyse the parameters which measure *how* the input is formed. For every of the three selected qualitative input parameters, we conduct at least 3 experiments with different values of these parameters, keeping the others constant. The results of the experiments can be found in Table 6.

|  |  | *Algorithms* | **DP** | **Projection** | **Greedy** | **GA** |
|---|---|---|---|---|---|---|
| ***Changing GCD (30 controls and 10 threats with 80 to 160 cost range)*** | **40 GCD** | *Execution time (sec)* | 418s | 74.6s | 0.03s | 3.8s |
|  |  | *Overall loss* | 627.1 | 627.1 | **700.6** | 627.1 |
|  | **20 GCD** | *Execution time (sec)* | 496s | 75.5s | 0.03s | 4.2s |
|  |  | *Overall loss* | 543.9 | 543.9 | **567.8** | 543.9 |
|  | **10 GCD** | *Execution time (sec)* | 1369.6s | 92.1s | 0.03s | 4.8s |
|  |  | *Overall loss* | 576.9 | 576.9 | **613.2** | 576.9 |
| ***Different range of cost (30 controls and 10 threats with 40 GCD)*** | **80 - 400 range** | *Execution time (sec)* | 6s | 1.2s | 0.01s | 3.6s |
|  |  | *Overall loss* | 706.6 | 706.6 | 706.6 | 706.6 |
|  | **80 - 160 range** | *Execution time (sec)* | 418s | 74.6s | 0.03s | 3.8s |
|  |  | *Overall loss* | 627.1 | 627.1 | **700.6** | 627.1 |
|  | **80 - 120 range** | *Execution time (sec)* | 1741.7s | 132.5s | 0.03s | 4.2s |
|  |  | *Overall loss* | 495.3 | 495.3 | **544.2** | 495.3 |
| ***Affected threats (30 controls and 10 threats with 40 GCD and 80 to 400 cost range)*** | **1 threat** | *Execution time (sec)* | >600s | 0.2s | 0.01s | 4.7s |
|  |  | *Overall loss* |  | 10277 | 10277 | 10277 |
|  | **4 threats** | *Execution time (sec)* | >600s | 3.3s | 0.01s | 3.8s |
|  |  | *Overall loss* |  | 2411.5 | 2411.5 | 2411.5 |
|  | **7 threats** | *Execution time (sec)* | 339.9s | 1.7s | 0.01s | 3.6s |
|  |  | *Overall loss* | 947.2 | 947.2 | 947.2 | 947.2 |
|  | **10 threats** | *Execution time (sec)* | 6s | 1.2s | 0.01s | 3.6s |
|  |  | *Overall loss* | 706.6 | 706.6 | 706.6 | 706.6 |

Table 6: Both time and accuracy of the solutions for different scenarios

We run the experiment with 30 controls and 10 threats, varying GCD (40, 20, and 10) used for the input (cost, in this case) generation. The cost of controls varies between 80 and 160. We see that GA is only slightly affected by the variation of GCD, but DP algorithms slow down quickly with the decrease of this parameter. The greedy algorithm does not change the time of its execution, as it does not depend on GCD. With these settings, we also see that the Greedy algorithm is not able to find the optimal answer.

We also analysed how the execution time depends on the variation of the control costs (see Table 6. Again, we see that DP-based algorithms significantly slow down as the variance decreases (the variant with projection slows down more slowly). GA also takes slightly more time once the variance becomes lower. Also, the Greedy algorithm slows slightly down, but its precision drops with a lower variance in control costs.

Finally, we let every control affect only a certain number of threats (e.g., 1, 4, 7, and 10). The original DP solution requires significantly more time if fewer threats affected, while GA and DP with projection solutions' time gradually drops. An interesting observation we have got for DP with projection: it finds the solution much faster for 1 threat affected case and immediately increases for 4 threats affected case. This can be concluded that 1 threat affected scenario is much simpler for the projection idea since it has more non-dominating vectors to project. For Greedy, it is almost constant around 0.01 seconds for all cases. In this experiment, we see that the input parameters are good enough for the Greedy algorithm to find the optimal answer.

## *5.4. Precision of GA and Greedy algorithms*

We see that GA and Greedy algorithms are faster than DP ones but they are imprecise. In other words, we cannot be sure that the result they produce is the optimal answer to the problem. Therefore, we would like to harden the input parameters for the approximate solutions to test their limits. Moreover, we are going to lower the settings for GA (limit number and population size) to test them.

| | | **Genetic Algorithm** | | | | | | | | | | **Greedy** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ***Limit number*** | **100** | | | **500** | | | **1000** | | | |
| | | ***Population size*** | ***100*** | ***500*** | ***1000*** | ***100*** | ***500*** | ***1000*** | ***100*** | ***500*** | ***1000*** | |
| **Increasing number of controls (20 threats 40 gcd with range of 80 to 400 costs)** | **20** | *Execution time (sec)* | 0.1 | 0.3s | 0.5s | 0.2s | 1.1s | 2s | 0.5s | 1.8s | 3.8s | 0.03s |
| | | *Overall loss* | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | **Success** |
| | **50** | *Execution time (sec)* | 0.1s | 0.3s | 0.6s | 0.3s | 1.2s | 2.3s | 0.5s | 2.2s | 4.2s | 0.2s |
| | | *Overall loss* | 8/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | **Failure** |
| | **100** | *Execution time (sec)* | 0.2s | 0.7s | 1.2s | 0.7s | 2.4s | 4.6s | 1.1s | 4.5s | 9s | 1.3s |
| | | *Overall loss* | 7/10 | 7/10 | 10/10 | 8/10 | 10/10 | 10/10 | 9/10 | 10/10 | 10/10 | **Failure** |
| **Increasing number of threats (50 controls 40 GCD with range of 80 to 400 costs)** | **20** | *Execution time (sec)* | 0.1s | 0.3s | 0.6s | 0.3s | 1.2s | 2.3s | 0.5s | 2.2s | 4.2s | 0.2s |
| | | *Overall loss* | 8/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | **Failure** |
| | **30** | *Execution time (sec)* | 0.2s | 0.8s | 1.4s | 0.7s | 2.6s | 5.4s | 1.2s | 5.9s | 10s | 0.31s |
| | | *Overall loss* | 5/10 | 10/10 | 10/10 | 8/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | **Failure** |
| | **40** | *Execution time (sec)* | 0.2s | 1.1s | 1.8s | 0.9s | 3.7s | 7.2s | 1.7s | 7.3s | 14s | 0.38s |
| | | *Overall loss* | 2/10 | 5/10 | 8/10 | 3/10 | 9/10 | 10/10 | 8/10 | 10/10 | 10/10 | **Failure** |
| **Different GCD (50 controls and 20 threats with range of 80 to 400 costs)** | **40 GCD** | *Execution time (sec)* | 0.1s | 0.3s | 0.6s | 0.3s | 1.2s | 2.3s | 0.5s | 2.2s | 4.2s | 0.2s |
| | | *Overall loss* | 8/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | **Failure** |
| | **20 GCD** | *Execution time (sec)* | 0.1s | 0.3s | 0.6s | 0.3s | 1.2s | 2.3s | 0.9s | 3.8s | 7.4s | 0.2s |
| | | *Overall loss* | 5/10 | 10/10 | 10/10 | 7/10 | 10/10 | 10/10 | 9/10 | 10/10 | 10/10 | **Success** |
| | **10 GCD** | *Execution time (sec)* | 0.1s | 0.3s | 0.6s | 0.3s | 1.2s | 2.1s | 0.6s | 2.2s | 4.1s | 0.21s |
| | | *Overall loss* | 5/10 | 9/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | **Failure** |
| **Different range of cost (50 controls and 20 threats with 40 GCD)** | **range of 80 - 400** | *Execution time (sec)* | 0.1s | 0.3s | 0.6s | 0.3s | 1.2s | 2.3s | 0.5s | 2.2s | 4.2s | 0.2s |
| | | *Overall loss* | 8/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | **Failure** |
| | **range of 80 - 240** | *Execution time (sec)* | 0.1s | 0.3s | 0.6s | 0.3s | 1.2s | 2.2s | 0.5s | 2.2s | 4.3s | 0.2s |
| | | *Overall loss* | 3/10 | 8/10 | 8/10 | 5/10 | 7/10 | 8/10 | 6/10 | 10/10 | 10/10 | **Failure** |
| | **range of 80 - 160** | *Execution time (sec)* | 0.1s | 0.3s | 0.6s | 0.3s | 1.2s | 2.2s | 0.5s | 2.3s | 4.2s | 0.2s |
| | | *Overall loss* | 3/10 | 4/10 | 7/10 | 5/10 | 5/10 | 8/10 | 8/10 | 10/10 | 10/10 | **Failure** |
| **Affected threats (50 controls and 20 threats with range of 80 to 400 costs and 40 GCD)** | **1 threat only** | *Execution time (sec)* | 0.2s | 0.8s | 1.5s | 0.7s | 3.1s | 6s | 1.3s | 6.1s | 22s | 0.03s |
| | | *Overall loss* | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | **Success** |
| | **3 threats affected** | *Execution time (sec)* | 0.2s | 0.6s | 1.3s | 0.6s | 2.8s | 5s | 1.1s | 5.1s | 18s | 0.06s |
| | | *Overall loss* | 5/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | 10/10 | **Failure** |
| | **7 threats affected** | *Execution time (sec)* | 0.1s | 0.5s | 0.9s | 0.5s | 1.9s | 3.6s | 0.9s | 3.5s | 7s | 0.09s |
| | | *Overall loss* | 4/10 | 7/10 | 10/10 | 6/10 | 10/10 | 10/10 | 8/10 | 10/10 | 10/10 | **Failure** |

Table 7: Different configurations for GA solution and their comparison with Greedy

Table 7 shows the result of different limits and population sizes for GA solution and their comparison with the Greedy results (binary representation of either *Success* or *Failure* for finding the optimal answer). Every experiment for GA settings has been conducted 10 times to evaluate the accuracy of the solution and the average execution time (highlighted and represented in seconds) is reported.

It is conceivable that with small population size and limit, GA performs much faster but fails more often. Also, other factors, i.e., GCD and Range of costs, affect the result of GA. Depending on the limit and population size as well as other factors, the gap between results is not huge. For instance, when we have a range of 80 - 160 costs with 100 limit and 100 population size, we see 4 different outcomes which difference are 29.4 (about 5% difference with the optimal answer) utmost.

The Greedy solution yields the result in a relatively shorter time as expected, yet fails to find the optimal solution in almost every experiment.

## *5.5. Analysis of results*

Our experiments show that our DP solution with projection really improves the speed of searching for the answer. The algorithm produces the results in a matter of 5-10 minutes for about 10-20 threats and for 50-100 controls. Since the decisions to be taken are strategic, i.e., will be valid for a long time (e.g., a year) we may see this time as acceptable. Moreover, the usage of a more powerful computing system will increase the speed of the proposed solutions. Furthermore, advanced methods for computation (e.g., parallel computations) will improve the speed even more.

Obviously, the exact algorithm cannot beat the approximate ones (although, we see that for a smaller quantity of input parameters, DP-based solutions are even faster than GA algorithms), but this is the price for assurance in the exact answer the algorithm produces. Naturally, if one considers the full set of controls from a standard (e.g., NIST [7] or ISO-27002 [52]) and the full set of threats (e.g., see ISO 27005 [18]) it is better to use GA. On the other hand, an organisation thinking about improving its self-protection is most likely to consider a moderate number of possible controls (i.e., 10-20) and be more focused on a moderate number of relevant threats (i.e., 10-20), thus, making application of our DP-based solution acceptable. Naturally, if the numbers are very small, an exhaustive search may be also applicable, but its time of execution grows too fast comparing to dynamic programming based algorithms.

Last, but not least, not only quantity but also the quality of the input has a great impact on the algorithms. Once GCD of the costs for security controls is high enough and their range is large, the DP-based solutions are able to terminate in a reasonable time. However, the reverse cases lead to much longer time to terminate. We also see that once only a few controls are affected, our solution with projection idea significantly outperforms the legacy one.

In sum, we may conclude that once a company has to select controls out of a limited number of alternatives, our solution is applicable in reality. On the other hand, with a larger number of controls, GA is more appropriate.

## 6. Discussion

In this section we would like to discuss some limitations of our work, most of which relate to its practical application.

Our solution is based on the competitive insurance market model, which is simple and widely used in theoretical insurance studies [55, 56]. This model allows us to focus on the core problems and simplifies the analysis. On the other hand, we acknowledge that in reality such a model should be adjusted to take into account various features deviating the result from the theoretical analysis. In case of insurance, such deviation is often modelled with additional loading factors [8], which increases the price to cover insurance expenses, ensures solvency, securing insurance risks, etc. In this paper, we do not consider how much the non-competitive market aspects affect the results of our analysis and leave this issue for future studies. We also would like to underline that other factors (e.g., insurance regulations, deducible, perception of risk, the interdependence of cyber security, etc.) may affect the results of our study and require specific analysis to estimate the deviation from our core model.

In this work, we assumed security controls' costs and probabilities to be independent of each other. In practice, some controls may require installation of others (e.g., security audit may require the presence of monitoring and logging mechanisms) or even conflict with each other (e.g., cryptography may reduce the effectiveness of security audit). Some of these issues can be resolved by pre-processing the input data (e.g., grouping some controls) and others can be resolved by some adjustments in the core algorithms. We leave these steps for future work.

Last but not least, we admit that our proposal relies on the knowledge of survival probabilities and expected losses, the values which are hard to find. The problem of identifying these values is well-known in the cyber security research and industrial community and is heavily related to the lack of statistical data available for research which is only becoming available [57]. On the other hand, we believe such values can be found by cyber insurers who are collecting data from their customers and are interested to use them for identification of these values. We also may see some real steps in this direction as IBM Security and Ponemon institute's Data Breach report [58] contains the study (and reports statistics) for the loss estimation depending on the security controls installed.

## 7. Conclusion and Future work

In this work, we theoretically analysed the problem of investments distributions for a risk-averse model of an organisation which is considering to buy cyber insurance. We have found that even in presence of several threats at the same time a competitive insurance market leads to full insurance coverage as optimal.

Also, our approach helps the organisation to identify the security controls required to be installed to optimise investments in self-protection and cyber insurance. The final problem we were solving aims for a cost-effective set of controls, in contrast to the usual state of the art approach aiming at selecting the controls with the total cost fitting the budget (and, thus, which could be not cost-efficient).

We provided an algorithmic solution which returns the exact answer to the main problem. The proposed solution proves to be faster than the direct application of the legacy approach our solution is based on. The solution is suitable for analysis of a reasonable set of available alternative controls (about 10-40) and threats (about 10), but for much larger sets of controls and threats genetic algorithm are more advisable to use. Although GA sometimes returns near optimal solutions, it is much more reliable than the Greedy algorithm and reasonably fast even with high settings.

## 8. Acknowledgement

This work was partially supported by projects H2020 MSCA NeCS 675320, H2020 MSCA CyberSure 734815, H2020 SPARTA 830892 and H2020 CyberSec4Europe 830929. We thank to anonymous reviewers.

## References

[1] Symantec: Internet Security Report. Volume 23. Available via `https://www.symantec.com/security-center/threat-report`, 2018

[2] Phil McCausland, Sam Petulla and Alastair Jamieson. Global Cyberattack Hits 150 Countries. Available via https://www.nbcnews.com/tech/internet/after-huge-global-cyberattack-countries-scramble-halt-spread-ransomware-n759121, 2017

[3] Cisco: Annual Cybersecurity Report. Available via `http://www.cisco.com/go/acr2017`, 2017.

[4] Ivan Homoliak, Flavio Toffalini, Juan Guarnizo, Yuval Elovici, and Martín Ochoa. "Insight into insiders and it: A survey of insider threat taxonomies, analysis, modeling, and countermeasures." ACM Computing Surveys (CSUR) 52, no. 2: pp. 1–40, 2019.

[5] Richard A.Caralli, James F.Stevens, Lisa R.Young, and William R.Wilson. Introducing octave allegro: Improving the information security risk assessment process. No. CMU/SEI-2007-TR-012. Carnegie–Mellon Univ Pittsburgh PA Software Engineering Inst, 2007.

[6] Miguel A.Amutio, Javier Candau, and José Manás. "Magerit-version 3, methodology for information systems risk analysis and management, book I-the method." Ministerio de administraciones públicas, 2014.

[7] NIST, Security and Privacy Controls for Federal Information Systems and Organizations, Tech. Rep. SP 800–53 Revision 4, National Institute of Standards and Technology, 2013.

[8] Angelica Marotta, Fabio Martinelli, Stefano Nanni, Albina Orlando, Artsiom Yautsiukhin. Cyber–insurance survey. Computer Science Review 24, pp. 35–61, 2017

[9] PartnerRe: Survey of Cyber Insurance Market Trends, Available via `https://partnerre.com/`, 2019.

[10] Rainer Böhme, Stefan Laube, and Markus Riek. "A fundamental approach to cyber risk analysis." Variance 12, no. 2: pp. 161–185, 2019.

[11] Savino Dambra, Leyla Bilge, and Davide Balzarotti. "SoK: Cyber Insurance—Technical Challenges and a System Security Roadmap." In 2020 IEEE Symposium on Security and Privacy (SP), pp. 293–309, 2020.

[12] Fabio Massacci, Joseph Swierzbinski, and Julian Williams. "Cyberinsurance and Public Policy: Self-Protection and Insurance with Endogenous Security Risks." In 16th Annual Workshop on the Economics of Information Security: WEIS 2017.

[13] Nikhil Shetty, Galina Schwartz, Jean Walrand: Can competitive insurers improve network security? In: A.Acquisti, S.Smith, A.R.Sadeghi (eds.) Proceedings of the 3$^{rd}$ International Conference on Trust and Trustworthy Computing,, Lecture Notes in Computer Science, vol. 6101, pp. 308–322. Springer, 2010.

[14] Galina Schwartz, Shankar Sastry: Cyber-insurance framework for large scale interdependent networks. In: Proceedings of the $3^rd$ International Conference on High Confidence Networked Systems, HiCoNS '14,. pp. 145–154. ACM, 2014.

[15] Marc Lelarge, Jean Bolot: Economic incentives to increase security in the internet: The case for insurance. In: Proceedings of the 28$^{th}$ IEEE International Conference on Computer Communications,. pp. 1494–1502, 2009.

[16] Ramaswamy Chandramouli. Security Strategies for Microservices-based Application Systems. No. Special Publication (NIST SP)-800-204, 2019.

[17] Ramaswamy Chandramouli, Murugiah Souppaya, and Karen Scarfone. NIST Guidance on Application Container Security. No. ITL Bulletin October 2017. National Institute of Standards and Technology, 2017.

[18] ISO/IEC, ISO/IEC 27005:2018 Information technology — Security techniques — Information security risk management (third edition), 2018.

[19] John J. Bartholdi III. "The knapsack problem." In Building intuition, pp. 19–31. Springer, Boston, MA, 2008.

[20] Andrew Fielder, Emmanouil Panaousis, Pasquale Malacaria, Chris Hankin, and Fabrizio Smeraldi. "Decision support approaches for cyber security investment." Decision support systems 86: pp. 13–23, 2016.

[21] Fabrizio Smeraldi, and Pasquale Malacaria. "How to spend it: optimal investment for cyber security." Proceedings of the 1$^{st}$ International Workshop on Agents and CyberSecurity. ACM, 2014.

[22] Maya Hristakeva, and Dipti Shrestha. "Solving the 0-1 knapsack problem with genetic algorithms." In Midwest instruction and computing symposium, 2004.

[23] Cristina Bazgan, Hadrien Hugot, and Daniel Vanderpooten. "Solving efficiently the $0-1$ multi-objective knapsack problem." Computers & Operations Research 36, no. 1: pp. 260–279, 2009.

[24] Anton Iliev, Nikolay Kyurkchiev, Asen Rahnev, and Todorka Terzieva. "Some models in the theory of computer viruses propagation." LAP LAMBERT Academic Publishing, 2019.

[25] Loren Paul Rees, Jason K. Deane, Terry R. Rakes, and Wade H. Baker. "Decision support for Cybersecurity risk planning." Decision Support Systems 51, no. 3: pp. 493–505, 2011.

[26] Megha Gupta. "A fast and efficient genetic algorithm to solve 0–1 knapsack problem." Int J Digit Appl Contemp Res 1, no. 6: pp. 1-5, 2013.

[27] Herbert A Simon. The sciences of the artificial. MIT press, 2019.

[28] Luca Allodi, and Fabio Massacci. "Security Events and Vulnerability Data for Cybersecurity Risk Estimation." Risk Analysis 37, no. 8: pp. 1606-1627, 2017.

[29] Humza Naseer, Graeme Shanks, Atif Ahmad, and Sean Maynard. "Towards an analytics-driven information security risk management: A contingent resource based perspective.", 2017.

[30] PwC: Global Cyber Insurance Survey. Available via `https://www.pwc.com/us/en/industries/insurance/library/cyber-insurance-survey.html`, 2018.

[31] ENISA: Incentives and barriers of the cyber insurance market in europe. Available via `http://www.goo.gl/BtNyj4on12/12/2014`, 2012.

[32] Hulisi Ogut, Nirup Menon, Srinivasan Raghunathan: Cyber insurance and it security investment: Impact of interdependent risk. In: Proceedings of the $4^{th}$ Workshop on the Economics of Information Security, 2005.

[33] Ruperto P.Majuca, William Yurcik, Jay P.Kesan.: The evolution of cyberinsurance. The Computing Research Repository pp. 1–16, 2006.

[34] Schneier Bruce. "Insurance and the computer industry." Communications of the ACM 44, no. 3, 2001.

[35] Isaac Ehrlich, Gary S.Becker: Market Insurance, Self-Insurance, and Self-Protection Foundations of Insurance Economics:, chap. Economics and Finance, pp. 164–189. Springer Netherlands, 1992.

[36] Arunabha Mukhopadhyay, Samir Chatterjee, Debashis Saha, Ambuj Mahanti, and Samir K. Sadhukhan. "Cyber-risk decision models: To insure IT or not?." Decision Support Systems 56: pp. 11-26, 2013.

[37] Chun-Jen Chung, Pankaj Khatkar, Tianyi Xing, Jeongkeun Lee, and Dijiang Huang. "NICE: Network intrusion detection and countermeasure selection in virtual network systems." IEEE transactions on dependable and secure computing 10, no. 4 : pp. 198-211, 2013.

[38] Tadeusz Sawik. "Selection of optimal countermeasure portfolio in IT security planning." Decision Support Systems 55.1: pp. 156-164, 2013.

[39] Rinku Dewri, Indrajit Ray, Nayot Poolsappasit, and Darrell Whitley. "Optimal security hardening on attack tree models of networks: a cost-benefit analysis." International Journal of Information Security 11, no. 3: pp. 167-188, 2012.

[40] Yunghee Lee, Tae Jong Choi, and Chang Wook Ahn. "Multi-objective evolutionary approach to select security solutions." CAAI Transactions on Intelligence Technology 2, no. 2: pp. 64-67, 2017.

[41] Iryna Yevseyeva, Vitor Basto-Fernandes, Michael Emmerich, and Aad van Moorsel. "Selecting optimal subset of security controls." Procedia Computer Science 64, pp. 1035-1042, 2015.

[42] Iryna Yevseyeva, Vitor Basto Fernandes, Aad van Moorsel, Helge Janicke, and Michael Emmerich. "Two-stage security controls selection." Procedia Computer Science 100, pp. 971-978, 2016.

[43] Valentina Viduto, Carsten Maple, Wei Huang, and David LoPez-PereZ. "A novel risk assessment and optimisation model for a multi-objective network security countermeasure selection problem." Decision Support Systems 53, no. 3: pp. 599-610, 2012.

[44] Valentina Viduto, Carsten Maple, Wei Huang, and Alexey Bochenkov. "A multi-objective genetic algorithm for minimising network security risk and cost." In 2012 International Conference on High Performance Computing & Simulation (HPCS), pp. 462-467. IEEE, 2012.

[45] Silvano Martello, David Pisinger, and Paolo Toth. "Dynamic programming and strong bounds for the 0-1 knapsack problem." Management Science 45, no. 3, pp. 414–424, 1999.

[46] Paolo Toth. "Dynamic programming algorithms for the zero-one knapsack problem." Computing 25, no. 1, pp. 29–45, 1980.

[47] Sami Khuri, Thomas Bäck, and Jörg Heitkötter. "The zero/one multiple knapsack problem and genetic algorithms." In Proceedings of the 1994 ACM symposium on Applied computing, pp. 188-193, 1994.

[48] Darrell Whitley. "A genetic algorithm tutorial." Statistics and computing 4, no. 2, pp. 65–85, 1994.

[49] Ahmad Hassanat, V. Prasath, Mohammed Abbadi, Salam Abu-Qdari, and Hossam Faris. "An improved genetic algorithm with a new initialization mechanism based on regression techniques." Information 9, no. 7, 2018.

[50] Nikolaj Goranin, and Antanas Cenys. "Genetic algorithm based Internet worm propagation strategy modeling under pressure of countermeasures." Journal of Engineering Science & Technology Review 2, no. 1, 2009.

[51] Suhail Owais, Vaclav Snasel, Pavel Kromer, and Ajith Abraham. "Survey: using genetic algorithm approach in intrusion detection systems techniques." In 2008 $7^{th}$ Computer Information Systems and Industrial Management Applications, pp. 300-307. IEEE, 2008.

[52] ISO/IEC 27002:2013 — Information technology — Security techniques — Code of practice for information security controls (second edition), 2013.

[53] Fabio Martinelli, Ganbayar Uuganbayar, and Artsiom Yautsiukhin. "Optimal Security Configuration for Cyber Insurance." In IFIP International Conference on ICT Systems Security and Privacy Protection, pp. 187-200. Springer, Cham, 2018.

[54] Alex J.Smola and Bernhard Scholkopf. Sparse greedy matrix approximation for machine learning. In Proceedings of the International Conference on Machine Learning, pp. 911–918, San Francisco, Morgan Kaufmann Publishers, 2000.

[55] Michael Rothschild and Joseph E.Stiglitz. Equilibrium in competitive insurance markets: An essay on the economics of imperfect information. The Quarterly Journal of Economics, 90(4): pp. 630–49, 1976.

[56] George A.Akerlof. "The market for “lemons”: Quality uncertainty and the market mechanism." In Uncertainty in economics, pp. 235-251. Academic Press, 1978.

[57] Mohammad Mahdi Khalili, Mingyan Liu, and Sasha Romanosky. "Embracing and controlling risk dependency in cyber-insurance policy underwriting." Journal of Cybersecurity 5, no. 1, 2019.

[58] Ponemon Institute and IBM: Cost of a Data Breach Report. Available via https://www.ibm.com/security/digital-assets/cost-data-breach-report/, 2020.

## 9. Appendix

*A. Indemnity*

Although it has already been proven in the literature that for a competitive market the optimal indemnity is equal to loss [8], we need to prove this also for a multi-threat scenario. In this regard, we apply Jensen's inequality for a concave function (for any concave function $\phi(t)$ $E[\phi(t)] \leq \phi(E[t])$) for Equation 9:

$$\sum_{\forall \vec{z}} p(\vec{z}|K_s, x) \cdot U(W^0 - (\vec{F} \odot \vec{p}(K_s|x)) \times \vec{I} - x + \vec{z} \times (\vec{I} - \vec{L})) \leq$$

$$U(\sum_{\forall \vec{z}} p(\vec{z}|K_s, x) \cdot \Big[W^0 - (\vec{F} \odot \vec{p}(K_s|x)) \times \vec{I} - x + \vec{z} \times (\vec{I} - \vec{L})\Big]) =$$

$$U(\left[\sum_{\forall \vec{z}} p(\vec{z}|K_s, x)\right](W^0 - x) - \left[\sum_{\forall \vec{z}} p(\vec{z}|K_s, x)\right] \cdot \Big[(\vec{F} \odot \vec{p}(K_s|x)) \times \vec{I}\Big] +$$

$$\left[\sum_{\forall \vec{z}} p(\vec{z}|K_s, x) \cdot \vec{z}\right] \times \vec{I} - \left[\sum_{\forall \vec{z}} p(\vec{z}|K_s, x) \cdot \vec{z}\right] \times \vec{L}).$$

Since $\sum_{\forall \vec{z}} p(\vec{z}|K_s, x) = 1$ and $\vec{F} \odot \vec{p}(K_s|x) = \sum_{\forall \vec{z}} p(\vec{z}|K_s, x) \cdot \vec{z}$, we get:

$$U(W^0 - x - \left[(\vec{F} \odot \vec{p}(K_s|x)) \times \vec{I}\right] + (\vec{F} \odot \vec{p}(K_s|x)) \times \vec{I} - (\vec{F} \odot \vec{p}(K_s|x)) \times \vec{L}) =$$
$$U(W^0 - x - (\vec{F} \odot \vec{p}(K_s|x)) \times \vec{L}).$$

The last part $(U(W^0 - x - (\vec{F} \odot \vec{p}(K_s|x)) \times \vec{L}))$ is the expected utility if $\vec{I} = \vec{L}$. In other words, Equation 9 is maximal if $\vec{I} = \vec{L}$.

## B. Algorithms

### B.1. Backtracking Algorithm

**Algorithm 3:** Recover selected controls

```
1  Function(BackTrack)
   Input: $x^*$, $c$, $T$, $C$
   Require:
2  $x^*$    // optimal investment found by searchForOptimalInvestments algorithm
3  $c$      // cost of controls
4  $T$      // auxiliary matrix T from searchForOptimalInvestments algorithm
5  $C$      // CGD for control cost
   Ensure: Optimal $K_s$
6  $K_s := \emptyset$
7  $x := x^*$      // we start with optimal investment level
8  for $i := 0$ to $n$ do
       // Iteration starts from the end of control's list
9      if $x - c[n - i - 1] < 0$ then
10         if $T[n - i - 1][x/C] \neq T[n - i][x/C]$ then
11             $K_s$.append(k[n - i])      // add $n - i$ -th control the selected controls list
12             $x := x$ - c[n - i - 1]      // Decrease the optimal investment by the cost of (n-i-1)-th control
   Return: $K_s$
```

### B.2. Greedy Algorithm

Function *GreedySelection* (lines 1 to 12) in algorithm 4 is the core function of the algorithm. It starts with an assumption that all controls are to be selected and gradually remove the ones which removal reduces the overall expenditure more than others. It stops when there is no control which removal decreases the expenditure. Auxiliary Function *Calc* (Algorithm 5) simply computes the overall expenditure with a set of currently selected controls; and auxiliary function *FindWorstControl* (Algorithm 6) selects the worst control to be removed.

**Algorithm 4:** The main function for Greedy approach

```
1  Function(GreedySelection)
   Input: $c$, $\pi$, $\vec{L}$, $\vec{p}_{init}$, $\vec{F}$
   Ensure: lowest minCost and optimalBudget for optimal security
           investment $x^*$
2  minCost, optimalBudget := Calc(c, $\pi$, $\vec{L}$, $\vec{p}_{init}$, $\vec{F}$)    // call Calc
    function to minCost and optimalBudget
3  index := FindWorstControl(c, $\pi$, $\vec{L}$, $\vec{p}_{init}$, $\vec{F}$)
4  while index $\neq$ -1 do
       // iterate until the loop ends
5      delete c[index]                    // Delete the cost of the worst control
6      delete $\pi$[index]     // Delete all probabilities of survival related to
        the worst control
7      currentCost, currentBudget := Calc(c, $\pi$, $\vec{L}$, $\vec{p}_{init}$, $\vec{F}$)    // call
        Calc function to currentCost and currentBudget
8      if currentCost < minCost then
               // check the condition and replace the minCost with currentCost
            if it meets
9          minCost := currentCost
10         optimalBudget := currentBudget
11     index := FindWorstControl(c, $\pi$, $\vec{L}$, $\vec{p}_{init}$, $\vec{F}$)
12 return minCost, optimalBudget
```

**Algorithm 5:** Calculation function of Greedy approach

```
1  Function(Calc)        // This function computes the minCost(expenditure)
   Input: $c$, $\pi$, $\vec{L}$, $\vec{p}_{init}$, $\vec{F}$
2  minCost := 0
3  for  i := 1 to length($\vec{L}$) do
4      prob := 1                                  // expected loss per threat i
5      for j := 1 to length(c) do
6          prob := prob $\cdot$ $\pi[j][i]$
7      minCost := $\vec{F}[i] \cdot \vec{L}[i] \cdot \vec{p}_{init}[i] \cdot$ prob          // put the sum of expected
        losses to the minCost
8  optimalBudget := sum(c)               // put sum of costs to opimalBudget
9  minCost := minCost + optimalBudget              // computes the overall
    expenditure
10 return minCost, optimalBudget
```

**Algorithm 6:** Removal of the worst control for Greedy approach

```
1  Function(FindWorstControl)        // function for finding worst controls
   Input: $c$, $\pi$, $\vec{L}$, $\vec{p}_{init}$, $\vec{F}$
   Ensure: Index of a control with the smallest risk reduction
2  $n$ := length(c)                  // define the length of costs
3  if $n == 1$ then
4      return $-1$
5  $h$ := length($\vec{L}$)          // define the length of probability of survival
6  $s$ := sum(c)                     // define the length of overall cost
7  $minCost, temp$ := Calc($c, \pi, \vec{L}, \vec{p}_{init}, \vec{F}$)    // call the Calc() function
8  $index$ := -1
9  for $j := 1$ to $n$ do
10     $val := 0$                    // the residual risk without j-th control
11     for $i := 1$ to $h$ do
12         prob := 1                 // probability of survival of threat i
13         for $l := 1$ to $n$ do
14             if $l \neq j$ then
15                 $prob := prob \cdot \pi[j][i]$    // add the relation of $\pi$
16             $val := val + \vec{F}[i] \cdot \vec{L}[i] \cdot \vec{p}_{init}[i] \cdot prob$
17     $currentCost$ := $val$ + s - c[i]    // compute the currentCost
18     if $currentCost < minCost$ then
19         $minCost := currentCost$          // update the minCost
20         $index$ := i                      // take the i-th control's index and return
21 return $index$
```

*B.3. Genetic Algorithm*

**Algorithm 7:** The main function of the GA algorithm

```
1  Function(GASelection)
   Input: $K$, $\pi$, $\vec{L}$, $\vec{F}$, $LIMIT$
   Require:
2  $LIMIT \in \mathbb{N}$                              // a constant value to run the GA
   Ensure: $lowest\ expenditure := (\vec{F} \odot \prod \pi(K_s|x)) \times \vec{L} + x$
3  generateRandomChromosomes($chromosomes$, $POP_{Num}$)    // generate initial population of chromosomes
4  sortByExpenditure(chromosomes)      // sort chromosomes by expenditure from lowest to largest
5  $theAnswer := chromosomes[0]$                 // Initializing theAnswer
6  for $i := 1$ to $LIMIT$ do
       $chromosomes := CrossOver(nextGenChromosomes)$          // Call CrossOver
       if $theAnswer.expenditure < chromosomes[0].expenditure$ then
           theAnswer := chromosomes[0]      // ensuring the best chromosome
10     $chromosomes := Mutation(nextGenChromosomes)$   // Call Mutation Function
11     if $theAnswer.expenditure < chromosomes[0].expenditure$ then
12         theAnswer := chromosomes[0]        // update the best chromosome
   Return: $theAnswer$
```

**Algorithm 8:** Merge Function for the GA

```
1  Function(Merge)
   Input: ch1, ch2, Y1, $n_k$
   Require:
2  $Y1 \in \mathbb{N}$                 // percentage of two-point CrossOver technique
3  $ch1$                                          // a chromosome in a pair
4  $ch2$                                    // another chromosome in a pair
5  $n_k \in \mathbb{N}$               // number of controls/genes in the chromosome
   Ensure: Merging two chromosomes ch1 and ch2
6  y = rand.range(0, 100)                            // some random percentage
7  if $y <= Y1$ then
8      l := rand.range(0, length(ch1.genes)-1)
9      r := rand.range(l, length(ch1.genes)-1)
10     for $i := l$ to $r$ do
11         ch1.genes[i] $\leftrightarrow$ ch2.genes[i]    // swap i-th genes of two chromosomes
12 else
13     for $i{:=}0$ to $n_k/2$ do
14         ch1.genes[i] $\leftrightarrow$ ch2.genes[i]    // swap i-th genes of two chromosomes
15 return $ch1$, $ch2$                          // Return a pair of chromosomes
```

**Algorithm 9:** CrossOver function for the New Population of the GA

```
Function(Crossover)
  Input: $chromosomes$, $x$, $N$, $LIMIT$, $Y2$, $Y3$, $b1$, $b2$
  Require:
  $chromosomes$                                   // a list of current chromosomes
  $N \in \mathbb{N}$                              // number of chromosomes in the population
  $b1 \in \mathbb{N}$                             // crossover percentage of good and good chromosomes
  $b2 \in \mathbb{N}$                             // crossover percentage of good and bad chromosomes
  $Y2 \in \mathbb{N}$                             // percentage of chromosomes for Elitism techniques
  $Y3 \in \mathbb{N}$                             // percentage of good chromosomes of the population
  Ensure: next Generation of Chromosomes obtained by CrossOver
  $nextGenChromosomes$ := []                      // Define nextGenChromosomes empty list
  E := $Y2 \cdot N/100$                           // Define the percentage of elitism in integer
  for $i := 1$ to $E$ do
      $nextGenChromosomes$.append(chromosomes[i])   // Add i-th chromosome without doing crossover
  r := $Y3 \cdot N/100$                           // turn the percentage of good chromosomes into integer
  for $i := E+1$ to $N$ do
      y := rand.range(0,100)                      // define a random integer
      if $y <= b1$ then
          // check if the crossover percentage of good chromosomes is greater than than a randomly chosen y
          ii := rand.range(0, r)                  // Define a random range of good chromosomes)
          jj := rand.range(0, r)
      else if $y <= b1 + b2$ then
          // check if the crossover percentage of good and bad chromosomes is greater than than a randomly chosen y
          ii := rand.range(0, r)
          jj := rand.range(r+1, N-1)              // Random range of bad chromosomes
      else
          // check if the crossover percentage of bad and bad chromosomes is greater than than a randomly chosen y
          ii := rand.range(r+1, N-1)
          jj := rand.range(r+1, N-1)
      ch1, ch2 := Merge(chromosomes[ii], chromosomes[jj])
      if $ch1.cost < x$ or $x = 0$ then
          // check if cost of ch1 is less than investment or equal to 0
          $nextGenChromsomes$.append(ch1)
      if $ch2.cost < x$ or $x = 0$ then
          // check if cost of ch1 is less than investment or equal to 0
          $nextGenChromsomes$.append(ch2)
  sort($nextGenChromosomes$)                      // Sort the chromosomes by expenditure
  Return: $nextGenChromosomes$
```

**Algorithm 10:** Mutation function for the New Population of the GA

```
Function(*Mutation*)
Input: *chromosomes*, $Y2$, $b3$
*chromosomes*                                  // a list of current chromosomes
$N \in \mathbb{N}$                                        // number of chromosomes in the population
$Y2 \in \mathbb{N}$                                      // percentage of chromosomes for Elitism techniques
$b3 \in \mathbb{N}$                                       // defined number of bits to mutate
Ensure: *next Generation* of *Chromosomes* (*nextGenChromosomes*)
        obtained by Mutation
*nextGenChromosomes* := []                    // Define *nextGenChromosomes* empty list
E := $Y2 \cdot N/100$                                 // Define the percentage of elitism as integer
for $i := 1$ to $E$ do
    *nextGenChromosomes*.append(*chromosomes*[i])     // update the list
for $i := E + 1$ to $N$ do
    ch := *chromosomes*[i]
    for $j := 1$ to $b3$ do
        y := rand.range(0, length(ch.genes)-1)        // reverse y-th gene in
        ch[i].genes[y] := (ch[i].genes[y] + 1) mod 2   // i-th chromosome
    *nextGenChromosomes*.append(ch)                   // update the list
sort(*nextGenChromosomes*)                    // Sort the chromosomes by *expenditure*
Return: *nextGenChromosomes*
```