ABSTRACT
This paper concerns using evolutionary algorithms for optimization in epidemics prevention. Spreading of the epidemic is simulated on a graph and the goal of optimization is to determine which vertices in the graph to vaccinate in order to minimize the number of vertices affected by the epidemic. Decisions whether to vaccinate a vertex or not are represented using a binary genotype.

In this paper an informed mutation operator is introduced. The presented operator uses machine learning approach for determining which positions in the genotype to flip. The learning model used in the paper is a neural network trained using graph-based features of the vertices (such as a vertex degree) and the information how often on average a given vertex is infected. Once trained, the learning model helps determine which positions in the current solution to mutate.

Results presented in the paper suggest that the proposed informed operator improves the ability of the evolutionary algorithm to produce good solutions to the tackled problem. Interestingly, a very good generalization was achieved. The model built using problem instances with 1000 vertices in the graph improved solutions for problem instances with up to 20000 vertices.

KEYWORDS
Knowledge-based optimization, Graph problems, REDS graphs, Epidemics Prevention

1 INTRODUCTION
Spreading threats in a graph is an abstraction that can be used for representing a number of real-life problems. The spreading threat can be a wildfire, an epidemic, a wave of company bankruptcies, a computer worm and others. The threat affects certain entities such as people, animals, companies or computers that can be represented by vertices in a graph. The threat spreads along edges in the graph which represent relationships between entities.

A common approach for applying optimization methods to such scenarios is to try to determine which vertices to protect in the graph in order to prevent the threat from spreading. In real life, protecting a vertex means, for example, vaccinating an individual, increasing a level of reserves in a company or reconfiguring a computer system and installing an anti-virus software. These actions incur a cost, but also diminish the chances that the threat will spread and thereby reduce the losses caused by illnesses, bankruptcies and computer system downtime.

In combinatorial optimization, the Firefighter Problem (FFP) is often studied, which was introduced by Hartnell in 1995 [7]. In the FFP spreading of fire is simulated on a graph in which the vertices can be in one of the states 'B' - burning, 'D' - defended and 'U' - untouched. Initially, a number $N_i$ of vertices in the graph are set on fire (the 'B' state) and subsequently spreading of fire is simulated in discrete time steps. In each time step a given number $N_f$ of vertices that are not yet burning (the 'U' state) can be defended by firefighters (set to the 'D' state), which prevents the fire from spreading to them. After that, the fire spreads along the edges of the graph from burning vertices to the unprotected ones. The goal of this optimization problem is to find in what order to protect the vertices in the graph ($N_f$ per a time step) in order to minimize the total number of burning vertices at the end of the simulation.

Apart from the FFP, other similar optimization problems were studied, for example a problem of protecting an economic system from system-wide failures [14]. In this problem the spreading of bankruptcies is simulated, which is governed by a more complex dynamic than in the FFP.

In this paper an informed mutation operator is proposed for solving graph-based optimization problems. Informed operators are one of the mechanisms for incorporation of a priori knowledge in evolutionary computation [8]. One of the approaches to designing informed operators is to use surrogate models [18]. This approach is particularly useful in engineering applications, in which solution evaluation can be costly if simulations are required to calculate the properties of the designed object based on the parameters encoded in the solution. Surrogate models can either be simplified models defined for a particular engineering problem [10], or can be general learning models than can be applied to various optimization problems [6]. Models used in the literature include statistical models [17] and models known from the machine learning field [5]. Informed operators are used to improve algorithmic aspects, such as population diversity [3], as well as to obtain better solutions to real-life problems [12].
2 PROBLEM DEFINITION
This paper addresses a problem of optimizing a vaccination scheme in order to improve epidemics prevention. While, in general, this problem is similar to the FFP discussed in the introduction, there are some differences which stem from the real-life aspects of the problem.

1 The threat spreading model. The spreading of the epidemic follows the SIRV (Susceptible, Infected, Recovered, Vaccinated) model \cite{11} with parameters \( \beta \) - the transmission probability and \( \gamma \) - the recovery probability. The spreading of the epidemic is non-deterministic (for \( \beta < 1 \), however, because 'I' individuals eventually transition to the 'R' state given enough time (for \( \gamma > 0 \) the epidemic is guaranteed to stop spreading in a finite number of steps.

2 Simulation-based evaluation. Following the simheuristic approach \cite{9} used in the paper on non-deterministic FFP \cite{15}, the spreading of the disease is simulated a number \( N_{\sim}\) of times and the average number of new cases of the disease (\( S \rightarrow I \) transitions) is calculated. Of course, in the classical FFP simulation is also used for determining the number of burnt vertices, but in the deterministic case \( N_{\sim} = 1 \) is sufficient, because for a given solution each simulation yields exactly the same result. In non-deterministic problems \( N_{\sim} \gg 1 \) is necessary (\( N_{\sim} = 100 \) is used in this paper), which makes the evaluations of solutions much more expensive.

3 Starting points. Similarly as in the FFP a number \( N_{v} \) of vertices are initially infected. However, in real life it is not possible to know in advance which individuals will fall ill at the beginning of an epidemic. Therefore in each simulation run \( N_{v} \) infected individuals are selected at random. Note, that such initialization prevents the optimizer from finding an easy solution that assumes protecting or surrounding the starting vertices.

4 When is the protection applied. In the FFP \( N_{f} \) vertices are protected in each time step. In this paper we assume that vaccination has to be performed before the epidemic breaks out in order for the individuals to build up immunity to the disease. Therefore, the vaccinated vertices become protected at the beginning of the simulation, that is, their state changes to 'V' and they are no longer susceptible to the disease. It is the goal of the optimization to determine which vertices to vaccinate.

5 Solution evaluation. In the FFP \( N_{f} \) vertices can be protected in each time step and the goal is to minimize the number of burnt vertices in the graph. In the presented problem the number of vaccinated vertices is not limited, but it is assumed that each vaccination costs \( C_{v} \) and each new case of the disease (a 'S' \( \rightarrow \) 'I' transition) costs \( C_{i} \). The optimization goal is to minimize the combined cost of vaccination and the cases of the disease.

Overall, the problem instance consists of a graph \( G = (V, E) \) and numerical parameters: \( N_{v} \) - the number of initially infected vertices, \( \beta \) - the transmission probability, \( \gamma \) - the recovery probability, \( C_{v} \) - the vaccination cost and \( C_{i} \) - the infection cost. Solutions are represented as binary vectors from the space \( \{0, 1\}^{N_{v}} \) (where \( N_{v} = |V| \) is the number of vertices in graph \( G \)), in which a coordinate equal to 1 denotes a vertex (individual) receiving the vaccine and 0 denotes one that does not receive the vaccine. Thus, for each solution \( x \in \{0, 1\}^{N_{v}} \) the number \( f_{\text{v}}(x) = \sum_{i = 1}^{N_{v}} : x[i] = 1 \) is the number of individuals that receive the vaccine. Evaluation of a solution \( x \) is calculated by performing \( N_{\sim} \) simulation runs, each starting with a different set of randomly selected, infected vertices.

At the beginning of each simulation run the vertices for which \( x[i] = 1 \) are set to the 'V' (Vaccinated) state and the remaining ones to the 'S' (Susceptible) state. Next, \( N_{v} \) vertices are independently, randomly selected with uniform probability distribution. Those selected vertices for which \( x[i] = 0 \) (the non-vaccinated ones) become infected (their state changes to 'I') and the vertices for which \( x[i] = 1 \) (the vaccinated ones) are unaffected. During the simulation run the number \( f_{i} \) of vertices that became infected (transited from the state 'S' to 'I') is recorded. The total cost of the solution \( x \) is then calculated as the value of \( f(x) = C_{v}f_{\text{v}} + C_{i}f_{i} \) averaged over all the simulation runs.

3 PROPOSED APPROACH
The general approach followed in this paper is to utilize knowledge about properties of elements of the problem representation (graph vertices) to make decisions how to modify solutions (vaccinate individual vertices or not) in order to obtain good optimization results. In this paper the knowledge is represented using a classification model able to determine which vertices in the graph to vaccinate and which not. Formally, we want to obtain a classifier \( \Psi : V \rightarrow \{0, 1\} \), where \( V \) - the set of vertices of graph \( G \) and the values in \( \{0, 1\} \) represent decisions whether to vaccinate vertex \( v \) (\( \Psi(v) = 1 \)) or not (\( \Psi(v) = 0 \)). Training of a classifier is a supervised machine learning process which requires training data consisting of the description of the classified objects and suggested "correct" answers (0 or 1 in our case). In this paper the training data set was extracted from the populations processed by an evolutionary algorithm solving problem instances with \( N_{v} = 1000 \) and the obtained learning model was used for solving problem instances with \( N_{v} \) up to 20000. Figure 1 presents the elements of the proposed method. The steps shown in the figure are discussed later in this section.

3.1 Attributes
Most classification algorithms assume that the objects are represented as a vector of attributes, often numerical, so that classified objects are actually points in \( \mathbb{R}^{d} \), where \( d \) is the number of attributes. In this paper vertices of graph \( G \) are described using four attributes presented in Table 1. The first three attributes are well-known measures used for characterizing graph vertices, and the last one contains information about the frequency with which a given vertex becomes infected. Based on these attributes a classifier \( \Psi : \mathbb{R}^{4} \rightarrow \{0, 1\} \) can be defined.

3.2 Training data
Training data for a supervised learning process can come from various sources, for example from solutions provided by domain experts or from automatically collected data sets. In this paper training data samples were collected during preliminary runs of the optimization algorithm on 30 problem instances with \( N_{v} = 1000 \)
we are more interested in when the elements of the solution vector vertices (for details of the test instances see Section 4.1). The average running time of each of these runs was 613.94 seconds on an Intel Xeon E5-2670 CPU.

Optimized solutions can be expected to have elements of the solution vector \( x[i] \) set to 1 for such vertices that are more important to protect (that is, such vertices which, if vaccinated, restrict the spreading of the disease effectively) and to 0 for such vertices that are less important to protect. Because the chosen attributes represent various measures of vertex centrality, their values can be expected to be useful for determining the influence of each vertex on the spreading of the disease. In the case of mutation operator we are more interested in when the elements of the solution vector should change their state. Therefore, in the proposed method the training sample is generated by flipping randomly selected elements in the solutions taken from the population during the run of the evolutionary algorithm and observing if an improvement of the optimization objective was achieved. The details of the sampling procedure are presented in Algorithm 1 which was run at the end of each EA generation.

The \( \text{SelectOneElement}() \) procedure selects one element from the given set with uniform probability distribution. As a result, a training sample \( S \) is collected containing pairs \( (a, c) \) where \( a \) is a vector of attributes discussed in 3.1 describing a graph vertex and \( c \) is a suggested value that should be selected for this vertex (0 - do not vaccinate, 1 - vaccinate). As shown in Algorithm 1 the class \( c = 0 \) is assigned when the selected solution element is 1 and flipping it to 0 improves the solution or when the selected solution element is 0 and flipping it to 1 does not improve the solution. Class \( c = 1 \) is assigned in the same manner, but, obviously, for the opposite outcomes obtained by changing the solution.

### 3.3 Classifier implementation and training

The function \( \Psi : \mathbb{R}^d \rightarrow \{0, 1\} \) is a general representation of a binary classifier working on objects represented using \( d \) numerical

### Table 1: Attributes of graph vertices used in the experiments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deg_entr</td>
<td>Degree centrality. The vertex degree (the number of edges that are adjacent to the vertex) divided by ( N_v - 1 ).</td>
</tr>
<tr>
<td>clos_entr</td>
<td>Closeness centrality. An inverse mean shortest path length from the vertex ( v ) to all the other vertices in the graph: ( (N_v - 1) \frac{1}{\sum_{w \neq v} d_{\text{min}}(v, w)} ), where ( d_{\text{min}}(v, w) ) is the length of the shortest path from ( v ) to ( w ).</td>
</tr>
<tr>
<td>betw_entr</td>
<td>Betweenness centrality. A measure indicating how often do the shortest paths between other vertices go through the vertex ( v ). Formally this measure is expressed as: ( \frac{1}{(N_v - 1)(N_v - 2)} \sum_{s \neq v, t \neq v} \frac{\sigma(s, t</td>
</tr>
<tr>
<td>inf_freq</td>
<td>Infection frequency. The fraction of simulation runs in which the vertex got infected.</td>
</tr>
</tbody>
</table>

### Algorithm 1: Generating training sample elements

**IN:**

- \( S \) - Already gathered training sample
- \( P \) - Current population of the EA
- \( A \) - Attributes of graph vertices discussed in 3.1

\( A[i] \) is a vector of attributes of the \( i \)-th vertex

**OUT:**

- \( S \) - The training sample with new examples

```
for x ∈ P do
    P₀ := \{i ∈ \{1, \ldots, N_v\} : x[i] = 0\}
    P₀ := SelectOneElement(P₀)
    a₀ := A[P₀]
    x₀ := x
    x₀[p₀] := 1
    if f(x₀) < f(x) then
        S := S ∪ (a₀, 1)
    else
        S := S ∪ (a₀, 0)
    P₁ := \{i ∈ \{1, \ldots, N_v\} : x[i] = 1\}
    P₁ := SelectOneElement(P₁)
    a₁ := A[P₁]
    x₁ := x
    x₁[p₁] := 0
    if f(x₁) < f(x) then
        S := S ∪ (a₁, 0)
    else
        S := S ∪ (a₁, 1)
```

Figure 1: Elements of the proposed method.
attributes. Obviously, a concrete model has to be selected as a working implementation of the function \( \Psi \). In this paper a three-layer perceptron was used with \( N_{in} = 4 \) neurons (corresponding to the attributes) in the input layer and \( N_{out} = 2 \) neurons (corresponding to the classes) in the output layer. The activation functions used in the network were the hyperbolic tangent \([19]\) in the hidden layer and the softmax function \([4]\) in the output layer which is a typical choice for neural classifiers with the number of outputs equal to the number of classes. Another neural network architecture with \( N_{out} = 1 \) and the logistic activation function in the output layer was considered (which is another common choice for binary classification), but in preliminary experiments it was found to be inferior to the former one and therefore was abandoned. The number of hidden neurons \( N_{hid} \) was tuned during parameter adjustment phase in \( [16] \) with the number of iterations set to \( N_{SCG} = 100 \) and 1000.

Note, that because of the selected network architecture the class encoding is not done using one output with 0 and 1 values, but instead with two-element output vectors \([1, 0]\) (corresponding to the “0” class) and \([0, 1]\) (corresponding to the “1” class). This is a typical encoding used for classification with neural networks with more than one output. With this encoding the network learns to output high values (close to 1) on output \( o_1 \) for one class and on \( o_2 \) output for the other class. Classifying an object is done by forwarding the attributes of the object through the network and comparing the outputs. A result \( o_1 > o_2 \) is interpreted as class “0” and \( o_1 < o_2 \) as class “1” (with \( o_1 = o_2 \) often considered a “don’t know” answer).

### 3.4 Informed mutation operator

The proposed mutation operator works in a similar manner as the classical bit flip mutation. That is, given \( n \) bits, each bit is randomly, independently flipped with a probability \( 1/n \). However, in the proposed operator a previously trained neural classifier is used for determining candidate positions of the bits to flip. When mutating a solution \( x \) the attributes describing the vertices of the graph (see section 3.1) are fed to the inputs of the neural network and, as discussed in section 3.3, for each vertex \( v_i \), \( i = 1, \ldots, N_v \) two values \( o_1 \) and \( o_2 \) are obtained at the outputs of the network. In the paper two modes of the working of the mutation operator were tested: **deterministic** (D) and **probabilistic** (P). In the deterministic mode the \( i \)-th position in the solution is added to the candidate set if the classification result is the opposite of the current state (that is when \( x[i] = 0 \) and \( o_2 > o_1 \), or when \( x[i] = 1 \) and \( o_2 < o_1 \)). In the probabilistic mode the probability of adding the \( i \)-th position to the candidate set is proportional to the normalized value of the “opposite” output. That is:

\[
P_{add}(i) = \begin{cases} 
\frac{o_2}{o_1 + o_2} & \text{when } x[i] = 0 \\
\frac{o_1}{o_1 + o_2} & \text{when } x[i] = 1
\end{cases} \tag{1}
\]

The working of the mutation operator proposed in this paper is presented in Algorithm 2. The \texttt{Random()} function draws a random number from the range \([0, 1]\) with uniform probability distribution and the \texttt{Net.Forward()} function calculates the output values of the network for the given inputs.

**Algorithm 2**: The working of the mutation operator proposed in this paper

<table>
<thead>
<tr>
<th>IN:</th>
<th>OUT:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Net ) - The neural network trained in sect. 3.3</td>
<td>( x ) - Mutated solution</td>
</tr>
<tr>
<td>( Mode ) - Working mode: deterministic (D) or probabilistic (P)</td>
<td></td>
</tr>
<tr>
<td>( x ) - The solution to mutate, ( x \in {0, 1}^{N_v} )</td>
<td></td>
</tr>
<tr>
<td>( A ) - Attributes of graph vertices discussed in sect. 3.1</td>
<td></td>
</tr>
<tr>
<td>( A[i] ) is a vector of attributes of the ( i )-th vertex</td>
<td></td>
</tr>
<tr>
<td>( \alpha_{p_{mut}} ) - A scaling factor for the mutation probability</td>
<td></td>
</tr>
</tbody>
</table>

**OUT**: \( x \) - Mutated solution

// Select candidate positions
\( M := \emptyset \)

for \( i := 1, \ldots, N_v \) do
  // Forward the attributes of the vertex through the network
  // The result is a two-element vector \([o_1, o_2]\)
  \([o_1, o_2] := \text{Net.Forward}(A[i])\)

switch \( Mode \) do
  // Deterministic mode
  case D do
    if \( (x[i] = 0 \text{ and } o_2 > o_1) \text{ or } (x[i] = 1 \text{ and } o_2 < o_1) \) then
      \( M := M \cup \{i\} \)
  // Probabilistic mode
  case P do
    if \( x[i] = 0 \) then
      \( P_{add} := \frac{o_2}{o_1 + o_2} \)
    else
      \( P_{add} := \frac{o_1}{o_1 + o_2} \)
    if \( \text{Random()} < P_{add} \) then
      \( M := M \cup \{i\} \)
// Bit-flip mutation at selected positions
for \( i := 1, \ldots, |M| \) do
  if \( \text{Random()} < \alpha_{p_{mut}} |M| \) then
    \( x[M[i]] := 1 - x[M[i]] \)

### 4 EXPERIMENTS AND RESULTS

The experiments presented in this paper were aimed at verifying if the proposed informed mutation operator improves the results obtained by the evolutionary algorithm solving the problem of optimizing a vaccination scheme in order to improve epidemics prevention.
4.1 Test instances

Test instances consist of a graph $G = (V, E)$ and several numerical parameters. In this paper REDS graphs were used [1] which combine spatial relationships with social synergy effects. These graphs were proposed in order to obtain a spatial edge distribution with denser cliques separated by relatively sparse areas, which resembles that of a real-life social network. Vertices in a REDS graph are randomly placed on a $[0, 1] \times [0, 1]$ square with the uniform probability distribution. The generation of edges is controlled by three parameters: the maximum distance between connected vertices $R$, the social energy $E$ and the synergy parameter $S$. Pairs of vertices placed no farther than $R$ apart are randomly selected and a connection is formed if the limit of the energy that each vertex can spend is not exceeded. The energy cost is proportional to the edge length $D_{ij}$, but it is discounted by the factor of $\frac{1}{1 + \sqrt{E}}$ for vertices $v_i$ and $v_j$ that have $k_{ij}$ neighbours in common, where $S$ is the synergy parameter. Note, that during edge generation this energy cost is calculated including the newly created edge, so it may happen that an edge is formed between vertices that have less than $D_{ij}$ energy left, thanks to the synergy effect and the resulting energy cost discount. For the tests graphs with $N_v = 1000, 1250, 1500, 1750, 2000, 2250, 2500, 5000, 10000, 15000$ and $20000$ were created. With the increasing number of vertices the distances between them decrease and the graph becomes very dense. To counteract this effect the parameters used for generating the graphs were adjusted in the following manner. For $N_v = 1000$ the parameters were set to $R = 0.1, E = 0.15$ and $S = 0.5$. For other values of $N_v$, the parameters were set to $R = 0.1\alpha, E = 0.15$ and $S = 0.5\alpha$, where $\alpha = \frac{1}{\sqrt{(N_v/1000)}}$. Thus, $R$ and $S$ were decreased proportionally to the square of the number of vertices and $E$ was kept constant. Figure 2 presents examples of the graphs obtained for $N_v = 1000$ and 5000. Clearly, even though the parameters of the graphs were corrected to account for the increasing number of vertices the differences in graph density are visible.

The problem definition includes several numerical parameters that control the epidemic dynamics and affect the calculation of solution cost. These parameters were set to the values presented in Table 2, the same for all test instances used in the experiments, regardless of the instance size.

![Figure 2: An example of REDS graphs obtained for $N_v = 1000$ and 5000 (colour online).](image)

4.2 Optimization algorithm

The optimization algorithm used in this paper was a classical genetic algorithm with an auto-adaptation mechanism [13] used for crossover operator selection. The crossover operators used in the experiments were the Single-Point crossover, the Two-Point crossover and the Uniform crossover. Because the evaluation of solutions is very expensive in the proposed problem, a stopping criterion based on the number of solution evaluations was used with the limit of $N_{SE} = 20000$ solution evaluations. The adopted limit translates to a running time several minutes for $N_v = 1000$ on a modern machine and, as shown in Figure 3, allows the algorithm to converge. For the parameters used in the experiments see Table 3 which describes the parameters and presents the values of these parameters obtained after tuning.

4.3 Parameter tuning

The tuning of parameters of the optimization algorithm was performed using the grid search approach. The tuned parameters and their ranges were: population size $N_{pop} = \{ 50, 100, 200, 500 \}$, crossover probability $P_{cross} = \{ 0.2, 0.4, 0.6, 0.8, 1.0 \}$ and mutation probability scaling factor $ap_{mut} = \{ 0.2, 0.5, 1.0, 2.0, 5.0 \}$. The mutation probability scaling factor was used to allow testing of mutation probabilities smaller and larger than the typical $1/n$ value used for the bit-flip mutation operator (where $n$ is the length of the mutated bit vector). For a given $ap_{mut}$ value the actual probability

![Figure 3: Convergence graph showing the minimum, average and maximum objective value in the population in one of the 30 optimization runs for $N_{SE} = 20000$, $N_v = 1000$, $N_{pop} = 100$, $P_{cross} = 1.0$ and $ap_{mut} = 2.0$.](image)

### Table 2: Numerical parameters of the test instances used in the paper

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of initially infected vertices</td>
<td>$N_i$</td>
<td>50</td>
</tr>
<tr>
<td>Transmission probability</td>
<td>$\beta$</td>
<td>0.1</td>
</tr>
<tr>
<td>Recovery probability</td>
<td>$\gamma$</td>
<td>0.1</td>
</tr>
<tr>
<td>Vaccination cost</td>
<td>$C_v$</td>
<td>1.0</td>
</tr>
<tr>
<td>Infection cost</td>
<td>$C_i$</td>
<td>10.0</td>
</tr>
</tbody>
</table>
of flipping each bit was $\alpha_{\text{mut}}/N_v$ in the case of the bit-flip mutation and $\alpha_{\text{mut}}/|M|$ (where $M$ is the set of candidate positions) in the case of the proposed informed mutation operator. Parameters of the evolutionary algorithm used in the paper are presented in Table 3.

### 4.4 Training data preparation

The evolutionary algorithm was run 30 times on 30 different training problem instances with $N_v = 1000$ vertices gathering training data using the method described in section 3.2. All the training examples from these runs were concatenated to a single training data set. Note, that this particular set of problem instances with $N_v = 1000$ was used solely for training and was not reused in the tests described in section 4.6 for which a separate set of 30 instances was generated.

### 4.5 Neural networks training

In the neural network training step two different values for the number of iterations of the Scaled Conjugate Gradient algorithm optimizing the network weights were used: $N_{\text{SCG}} = 100$ and 1000. The classification quality was measured using four quality measures $Q$: accuracy (ACC), precision (P), true positive rate (TP), a.k.a. recall and true negative rate (TN, a.k.a. specificity). For each pair of the $N_{\text{SCG}}$ setting and quality measure $Q$ the following procedure was used to select the best neural network:

- The number of neurons in the hidden layer was set to $N_{\text{hid}} = 1, 2, \ldots, 30$.
- Because the a priori probabilities of the classes are not necessarily equal in the solved problem, three settings for class balancing $B$ were tested: none (no balancing), downsample (take fewer examples from the majority class) and oversample (take multiple copies of examples from the minority class).
- For each value of $N_{\text{hid}}$ and each class balancing setting $B$ a 10-fold cross-validation procedure [2] was performed, training 10 neural networks and producing 10 values of the measure $Q$.
- The best settings for the number of neurons in the hidden layer $N_{\text{hid}}^*$ and class balancing $B^*$ were selected based on the average value $Q$ of the quality measure $Q$ calculated from the 10 cross-validation folds.

- The best neural network (with respect to the classification quality measure $Q$) was selected from the 10 cross-validation folds performed for the best found values $N_{\text{hid}}^*$ and $B^*$.

Overall, eight different neural networks were selected (for two different values of $N_{\text{SCG}}$ and for four different classification quality measures). The results of this experimental step are presented in Table 4. Reported running times were obtained when training the networks using a CUDA-based implementation on a computer with the Intel i7-6700HQ CPU running at 2.6 GHz with an NVIDIA GeForce 960M GTX graphics card (GPU).

### 4.6 Testing of the informed mutation operator

After the neural networks are trained, they can be used in the informed mutation operator according to Algorithm 2. In the experiments the informed mutation operator was tested in two working modes (deterministic and probabilistic) using neural networks trained with two settings of the number of iterations of the weights optimization algorithm $N_{\text{SCG}} = 100$ and 1000. Networks that performed best with respect to each of the four classification quality measures were tested. Networks selected using the accuracy (ACC), precision (P) and true negative rate (TN) were found not to produce competitive results. However, when the informed mutation operator used the networks selected with respect to the best true positive rate (TP), results competitive to the non-informed operator were obtained. These two neural networks are Net(100, TP) and Net(1000, TP) in Table 4. Table 5 presents the results obtained using the proposed informed mutation operator with various settings and, for comparison, the non-informed operator (the classical bit-flip mutation). Presented values are medians of the best solution evaluation obtained using each method in 30 independent runs on 30 different test instances. Because the tackled problem is a problem of minimizing costs, lower values are better. Values obtained using the informed mutation operator that are better (lower) than the ones obtained using the non-informed operator are marked in bold in the tables.

In order to verify the significance of the observed results, statistical testing was performed in which the results produced by the informed operator were compared to those produced by the non-informed one. Because the normality of the distribution of the observed values cannot be guaranteed the Wilcoxon signed rank test was used which does not require the normality assumption and has a null hypothesis which states the equality of medians. In Table 5 the p-values obtained in this test are presented. The results obtained using the informed operator that are significantly better at the significance level $\alpha = 0.05$ are underlined.

For the informed mutation operator working in the probabilistic mode and using the Net(1000, TP) network (the last column in the table) results for all the test instances are in favour of the proposed method and are statistically significant (p-values lower than $10^{-4}$ in each case). Therefore, for these results the Family-Wise Error Rate (FWER) was calculated which is the probability that in the entire series of experiments for different values of $N_v$ at least one conclusion stating the statistical significance is actually false. This FWER value was calculated as $1 - \prod_i (1 - p_i)$, where each $p_i$ is a p-value obtained in a comparison made for a given value of $N_v$ (one row in the results table). The FWER value obtained for the results...
Table 4: Parameters of the trained neural networks

<table>
<thead>
<tr>
<th>Network name</th>
<th>$N_{SCG}$</th>
<th>Quality measure $Q$</th>
<th>Class balancing</th>
<th>$N_{hid}^*$</th>
<th>Average quality measure $Q$</th>
<th>Quality measure for the best network</th>
<th>Training time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net(100, ACC)</td>
<td>100</td>
<td>accuracy</td>
<td>none</td>
<td>19</td>
<td>0.53441</td>
<td>0.60449</td>
<td>117.09</td>
</tr>
<tr>
<td>Net(100, P)</td>
<td>100</td>
<td>precision</td>
<td>oversample</td>
<td>17</td>
<td>0.46935</td>
<td>1.00000</td>
<td>140.40</td>
</tr>
<tr>
<td>Net(100, TN)</td>
<td>100</td>
<td>true negative rate</td>
<td>oversample</td>
<td>26</td>
<td>0.69122</td>
<td>1.00000</td>
<td>140.00</td>
</tr>
<tr>
<td>Net(100, TP)</td>
<td>100</td>
<td>true positive rate</td>
<td>oversample</td>
<td>11</td>
<td>0.66953</td>
<td>1.00000</td>
<td>113.43</td>
</tr>
<tr>
<td>Net(1000, ACC)</td>
<td>1000</td>
<td>accuracy</td>
<td>none</td>
<td>10</td>
<td>0.55316</td>
<td>0.60265</td>
<td>816.67</td>
</tr>
<tr>
<td>Net(1000, P)</td>
<td>1000</td>
<td>precision</td>
<td>downsample</td>
<td>30</td>
<td>0.47077</td>
<td>1.00000</td>
<td>97.32</td>
</tr>
<tr>
<td>Net(1000, TN)</td>
<td>1000</td>
<td>true negative rate</td>
<td>none</td>
<td>10</td>
<td>0.76559</td>
<td>1.00000</td>
<td>816.67</td>
</tr>
<tr>
<td>Net(1000, TP)</td>
<td>1000</td>
<td>true positive rate</td>
<td>downsample</td>
<td>8</td>
<td>0.68104</td>
<td>1.00000</td>
<td>79.37</td>
</tr>
</tbody>
</table>

Table 5: Median results obtained by the mutation operators tested in the experiments in the limit of $N_{SE} = 20000$ solution evaluations.

<table>
<thead>
<tr>
<th>Instance size $N_v$</th>
<th>Non-informed operator</th>
<th>deterministic</th>
<th>probabilistic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$N_{SCG} = 100$</td>
<td>$N_{SCG} = 1000$</td>
<td>$N_{SCG} = 100$</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>p-value</td>
<td>Median</td>
</tr>
<tr>
<td>1000</td>
<td>903.0</td>
<td></td>
<td>910.5</td>
</tr>
<tr>
<td>1250</td>
<td>1110.0</td>
<td></td>
<td>1177.5</td>
</tr>
<tr>
<td>1500</td>
<td>1319.0</td>
<td></td>
<td>1316.0</td>
</tr>
<tr>
<td>1750</td>
<td>1545.0</td>
<td></td>
<td>1522.5</td>
</tr>
<tr>
<td>2000</td>
<td>1780.0</td>
<td></td>
<td>1738.0</td>
</tr>
<tr>
<td>2250</td>
<td>2020.0</td>
<td></td>
<td>1943.0</td>
</tr>
<tr>
<td>2500</td>
<td>2280.0</td>
<td></td>
<td>2165.0</td>
</tr>
<tr>
<td>5000</td>
<td>6319.0</td>
<td></td>
<td>4837.3</td>
</tr>
<tr>
<td>10000</td>
<td>36078.5</td>
<td></td>
<td>2936.1</td>
</tr>
<tr>
<td>15000</td>
<td>64055.0</td>
<td></td>
<td>60514.5</td>
</tr>
<tr>
<td>20000</td>
<td>89480.0</td>
<td></td>
<td>86534.0</td>
</tr>
</tbody>
</table>

The presented in the table is $4.076 \cdot 10^{-4}$ which confirms that it is very unlikely that even one of the results obtained using the informed operator is not statistically different from the corresponding result obtained using the non-informed operator.

4.7 Running time of the algorithms

This section compares running times of the optimization algorithms using the informed mutation operator and the non-informed one. The proposed mutation operator uses neural networks for determining which elements in the solution to mutate. Forwarding the attributes through a neural network is much faster than network training, however, it still requires a considerable number of mathematical operations. It may seem that the proposed mutation operator must increase the running time of the optimization algorithm. However, comparison of running times presented in Table 6 shows that it is not so. In the table the running time of the algorithm using the non-informed mutation operator is compared to the running time of the algorithm using the informed mutation operator working in the probabilistic mode and using the Net(1000, TP) network. Better of the two running times for each instance size $N_v$ is underlined.

It turns out that in 9 out of 11 cases the algorithm using the informed operator was actually faster. This observation may seem very surprising at first, but it is easily explained by the fact that if an algorithm finds better solutions, these solutions evaluate faster, because the spreading of the epidemic is contained at earlier steps, on average. Therefore, the proposed method not only produces competitive results given a limited number of solution evaluations (Table 5), it is also able to shorten the running time of the optimization process (Table 6).

5 CONCLUSIONS

In this paper an informed mutation operator was proposed, which uses the machine learning approach for deciding which bits in a solution vector to consider for mutation. The learning model is a neural classifier trained on data collected during a preliminary run of the optimization algorithm. Several observations have been made in the paper, namely:
The topic introduced in this paper seems worthy of further investigation. Some of the ideas that could be developed in continuation of this work are: using other classification models than neural networks, extending ideas presented in this paper to other operators such as crossover or local search and developing similar mechanisms for multiobjective optimization.

Table 6: Comparison of running times of the algorithms (in seconds, means from 30 runs) using the non-informed mutation operator and the informed one (shorter of each two is underlined).

<table>
<thead>
<tr>
<th>Instance size $N_v$</th>
<th>Non-informed operator</th>
<th>Informed operator, probabilistic mode, Net(1000), TP network</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>371.30</td>
<td>343.02</td>
</tr>
<tr>
<td>1250</td>
<td>480.20</td>
<td>497.95</td>
</tr>
<tr>
<td>1500</td>
<td>684.22</td>
<td>645.26</td>
</tr>
<tr>
<td>1750</td>
<td>896.14</td>
<td>849.70</td>
</tr>
<tr>
<td>2000</td>
<td>1218.75</td>
<td>1171.57</td>
</tr>
<tr>
<td>2250</td>
<td>1478.95</td>
<td>1419.00</td>
</tr>
<tr>
<td>2500</td>
<td>1880.12</td>
<td>1750.25</td>
</tr>
<tr>
<td>5000</td>
<td>7685.30</td>
<td>7122.31</td>
</tr>
<tr>
<td>10000</td>
<td>26460.84</td>
<td>27716.64</td>
</tr>
<tr>
<td>15000</td>
<td>42299.57</td>
<td>42209.42</td>
</tr>
<tr>
<td>20000</td>
<td>59011.98</td>
<td>56934.01</td>
</tr>
</tbody>
</table>

- The informed mutation, in general, works better than the non-informed one, especially when working in the probabilistic mode using neural networks trained for $N_{SCG} = 1000$ iterations. The results for all the instance sizes $N_v$ were in favour of the informed method and were verified to be statistically significant. Also, the Family-Wise Error Rate (FWER) calculated over all the tests performed for various instance sizes is low (below 0.001), which confirms that, with high confidence, the informed operator can be assumed to be better than the non-informed one.
- It is possible to extract knowledge from optimization process running for smaller instances ($N_v = 1000$) and to reuse this knowledge for much larger instances. In the experiments instances with up to $N_v = 20000$ vertices were solved using the same classifier trained while solving the problem instances with $N_v = 1000$ and the proposed method still had an advantage over the non-informed approach.
- Neural networks selected using the true positive rate (a.k.a. recall) as the classification quality measure clearly outperformed other ones. This measure indicates how well a classifier is able to point out the objects from the positive class (in this paper: the vertices to vaccinate). The observed results suggest that it is most important to identify all the vertices that indeed require protecting and it is less important to avoid protecting too many vertices.
- The ability of the algorithm to find better solutions allows shortening of the running time, because in better solutions the epidemic is stopped earlier and simulations take shorter to complete.

ACKNOWLEDGMENT
This work was supported by the Polish National Science Centre under grant no. 2015/19/D/HS4/02574. Calculations have been carried out using resources provided by Wroclaw Centre for Networking and Supercomputing (http://wcssl.pl), grant No. 407.

REFERENCES