Automatika

Journal for Control, Measurement, Electronics, Computing and Communications

ISSN: 0005-1144 (Print) 1848-3380 (Online) Journal homepage: https://www.tandfonline.com/loi/taut20

Optimizing physical protection system using domain experienced exploration method

Dejan Čakija, Željko Ban, Marin Golub & Dino Čakija

To cite this article: Dejan Čakija, Željko Ban, Marin Golub & Dino Čakija (2020) Optimizing physical protection system using domain experienced exploration method, Automatika, 61:2, 207-218, DOI: 10.1080/00051144.2019.1698192

To link to this article: https://doi.org/10.1080/00051144.2019.1698192

Published online: 09 Dec 2019.

AUTOMATIKA
2020, VOL. 61, NO. 2, 207–218
https://doi.org/10.1080/00051144.2019.1698192

REGULAR PAPER

OPEN ACCESS Check for updates

Optimizing physical protection system using domain experienced exploration method

Dejan Čakija[a], Željko Ban[a], Marin Golub[a] and Dino Čakija[b]

[a]Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia; [b]Faculty of Transport and Traffic Sciences, University of Zagreb, Zagreb, Croatia

ABSTRACT
Assessing physical protection system efficiency is mostly done manually by security experts due to the complexity of the assessment process and lack of tools. Computer aided numerical vulnerability analysis has been developed to quantitatively assess physical protection systems. A variety of methods have been proposed to optimize physical protection systems, where one of the most advanced approaches entails precisely defining which security components should be selected and where they should be placed at protected facilities, taking into consideration adversary type, to maximize the probability that an adversary will be stopped at minimum system cost. The most computationally intensive part of the optimization process is the evaluation. The evaluation involves recreating search space and finding optimal adversary's attack paths from each entry point. We present the domain experienced exploration method that optimizes evaluation process during the search for optimum solutions, considering results from previous evaluations. Performed experiments show that using the presented method, in real-world domains, results in a reduction of evaluation iterations.

ARTICLE HISTORY
Received 5 July 2019
Accepted 4 November 2019

KEYWORDS
Multi-objective optimization of physical protection system; domain experienced exploration; genetic algorithms; PPS design; numerical vulnerability analysis

1. Introduction

The basic function of any physical protection system is to detect and delay an adversary's attack so that security team can prevent the adversary from reaching the intended target, such as people, valuable items and proprietary data [1]. An example of highly valuable target is data centre whose destruction or sabotage causes possible downtime and the extraction or loss of data. For that reason, beside energy management, event monitoring and effective maintenance, security system is a key element of a highly available and secure data centre [2–4]. The primary components of physical protection systems are security cameras, intrusion detection system, video analytics, locks, vaults, monitoring application, etc. Selecting security components and their placement is part of physical protection system planning and is typically executed by security analysts, based mostly on experience. A disadvantage of this approach is that the effectiveness of the physical protection system is assessed subjectively. An alternative approach is to use computer aided numerical vulnerability analysis to quantitatively assess physical protection systems. The main concept of quantitative methods is to find possible physical paths that an adversary could use to reach the goal and to detect which paths give the adversary the best probability of a successful attack. Once detected, these paths are considered the most critical paths.

One of the earliest papers on the numerical assessment of physical protection system [5] uses stochastic modelling. A facility is presented as block diagrams and networks and time that an adversary requires to complete a task on an attack path and guard time are expressed as probability density functions in the complex domain, using Laplace transforms. Sandia National Laboratories is very active in developing quantitative assessment methods. In the 1970s Sandia had already developed a number of quantitative assessment methods, including the Estimate of Adversary Sequence Interruption (EASI) method that is used as the foundation for many other assessment methods [1,6]. Assessing protection systems requires that a facility is modelled in a format appropriate for numerical vulnerability analysis. Systematic Analysis of Vulnerability to Intrusion (SAVI) [1] is a method that expects the operator to define possible adversary paths as adversary sequence diagram (ASD) – simple model where a facility is split in several adjacent physical areas, between each are protection layers, containing path elements, interconnected via path segments. An improvement over this kind of model representation is SAPE – Systematic Analysis of Physical Protection Effectiveness [7], where the facility layout is split into a two-dimensional, grid-like map with a certain resolution. This kind of representation is

CONTACT Dejan Čakija dejan.cakija@gmail.com

computationally-demanding due to the large number of elements required for the model.

More advanced techniques, that allow automatization of assessment process, include methods where a facility is defined in 3D applications, usually BIM – Building Information Modeling, and then transferred into the graph-based model, where nodes represent facility building elements and edges are available paths between them [8–10].

Finding critical paths in complex search space is computationally expensive. Heuristic methods have an advantage in this area as they decrease the number of searches [11]. Informed search algorithms use a heuristic function that guides the search space exploration based on problem-specific knowledge. Besides the well-known heuristic algorithm A* [12,13], modern computer architectures allow running A* in parallel, using algorithms such as PNBA* [14] and Meet in the middle (M2M) [15] that starts parallel search from start and end nodes and finishes in the middle of graph when all requirements are met. Another approach in the search for a critical path is using an ant colony optimization algorithm [16].

Previously we presented tools to asses existing or proposed physical protection systems. The next step is to optimize the facility protection by placing security elements on critical paths. In this scenario, security experts attempt to identify optimal tradeoffs between system cost and probability of adversary interruption. The optimization process is presented in Figure 1.

Different approaches have been proposed to help security experts with system optimization. Some authors use genetic algorithms for optimizing the coverage set (percentage values) of protection mechanisms at available budget [17,18], or model-based methodology with Petri Net patterns [19]. More detailed optimization requires that method provide the exact selection of security components and their placement for achieving near-optimal solution. A significant challenge in the optimization process is that there are many possibilities for selecting security components and their placement. If a security expert makes selections based solely on experience, there is a high probability that the selected combination will be far from an optimal solution. Complex facilities and a variety of security elements offer too many combinations for a human to find near-optimal solutions, without using appropriate tools. One method for solving such problems is using evolutionary algorithms. Game-theoretic model has been presented [20] that, besides considering security elements properties and uncertainty of response force times, considers budget limitations and the impact of false alarms on system performance. Authors use greedy algorithm to seed the genetic algorithm. This model is extended with support for connecting between weather, visibility conditions and intruder capabilities with detection probability, and interruption probability is calculated including the impact of nuisance and false alarm rates on operator performance [21]. Some of the listed articles expect predefined budget or optimize only one starting point or optimize predefined critical paths, without taking into consideration that applying security equipment changes critical paths.

Our approach considers all available starting points for an adversary when optimizing protection systems. Also, before each search iteration for critical paths, it redefines search space based on selected security components so that calculated probability interruption is always correct, and not approximated. The next chapter defines a model that is used for automatically finding near-optimal solutions of physical protection systems.

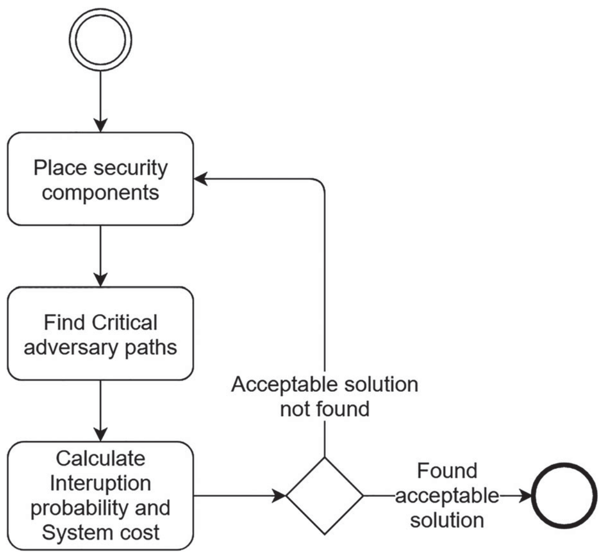

Place security components
Find Critical adversary paths
Calculate Interuption probability and System cost
Acceptable solution not found
Found acceptable solution

Figure 1. Optimization process.

2. Definitions

Let $G = \{\mathcal{V}, \mathcal{E}\}$ be a graph where $\mathcal{V}$ is a set of vertices, $|\mathcal{V}| = n$, and $\mathcal{E}$ is the set of edges $|\mathcal{E}| = m$. Each vertex is mapped to the element of facility that is being protected, such as door, window or room. Vertex has following properties:

$dp(v)$ – detection probability.
$pt(v)$ – time required for passing the element,
$sn(v)$ – standard deviation of time required for passing the element.
$ae(v)$ – set of security elements available for applying to the facility element
$ap(v)$ – set of applied security elements to the facility element
$co(v)$ – cost of applied security elements to the facility element

Let $\mathcal{S}$, such that $\mathcal{S} \subset \mathcal{V}(G)$, be a set of vertices that could be used by an adversary as starting attack points and let $\mathcal{T}$, such that $\mathcal{T} \subset \mathcal{V}(G)$, be a set that contains vertex which is attacker target.

The edges represent possible movement of the adversary from one vertex to the other. Graph is directed, edges are defined as $e_{ij} = (v_i, v_j)$, where $e_{ij} \in \mathcal{E}$.

Edge has following properties:

$pt(e_{ij})$ – time required for passing from vertex v_i to vertex v_j,

$sn(e_{ij})$ – standard deviation of time required for passing from vertex v_i to vertex v_j,

For each element of set $\mathcal{S}$, we could find a path

$$p_i := (v_1 e_{12} v_2 \dots v_{N-1} e_{N-1N} v_N) \tag{1}$$

so that

$$f(p_i) = f(v_1 e_{12} v_2 \dots v_{N-1} e_{N-1N} v_N) = \min(P_I) \tag{2}$$

where P_I is probability that adversary would be detected and intercepted before reaching the target vertex $v_t \epsilon \mathcal{T}$.

Interruption probability can be calculated as

$$P_I = P_{R1} P_{D1} P_C + \sum_{i=2}^{N} P_{Ri} P_{Di} P_c \cdot \prod_{j=1}^{i-1} (1 - P_{Dj} P_c) \tag{3}$$

where P_{Ri} is a probability that security guards will arrive from vertex v_i to target vertex v_t before adversary, P_{Di} is a probability that adversary will be detected at v_i, and P_C is a probability that communication is successful during detection and response.

Probability detection P_{Di} is a function of independent detection probability of all security elements applied at vertex $\mathrm{v_i}$ and equals $dp(v_i)$.

Probability function P_{Ri} is calculated as:

$$P_{Ri} = E(T_{ui} > 0, s_n) = \int_0^\infty \frac{1}{s_{ni}\sqrt{2\pi}} e^{\frac{(t-T_{Ui})^2}{2s_{ni}^2}} \mathrm{d}t \tag{4}$$

T_{Ui} is difference between time required for guard to reach v_t after adversary is detected T_G, and T_{Ri} – time required for adversary to reach v_i from v_1. T_{Ri} equals $\sum_{j=1}^{i} pt(x)$, where x is element of path $(v_1 e_{12} v_2 \dots v_{i-1} e_{i-1i} v_i)$.

$$T_{Ui} = T_G - T_{Ri} \tag{5}$$

Standard deviation s_n of T_{Ui} equals

$$s_{ni}^2(T_{Ui}) = s_{ni}^2(T_{Ri} - T_G) = s_n^2(T_{Ri}) + s_n^2(T_G) \tag{6}$$

System cost can be calculated as

$$cost(G) = \sum_{i=1}^{n} co(v_i), \text{where } v_i \in \mathcal{V} \tag{7}$$

3. Optimization problem

The objective is to find which security elements from ae(v) should be applied to facility elements, so that for minimal cost of security system $cost(G)$, a maximum probability that adversary is stopped P_I is achieved. This is a multi-objective problem, where number of possible solutions for each adversary type is $|\mathcal{S}| \cdot 2^m$, having $m = \sum_{i=1}^{n} |ae(v_i)|$ where $n = |\mathcal{V}|$.

The goal is to find a set of non-dominated solutions within an entire feasible search space.

$$P^* = \{x \in \Omega | \neg \exists x' \in \Omega \mathrm{F}(x') \leq F(x)\} \tag{8}$$

Such set is called the pareto-optimal or non-dominated set.

$$f(x) = (F(x), cost(x)) \tag{9}$$

Figure 2 shows set of solutions where Pareto-optimal solutions are presented as rounded marks, while non-optimal solutions are presented as squares.

F is a fitness function and represents average value of probability that adversary will be successful in his attack. If P_I equals 0, then penalty is introduced so that optimization algorithm finds solutions where $P_I = 0$ is omitted.

$$F(x) = \frac{\sum_{i=1}^{|\mathcal{S}|} f(x_i) \begin{cases} P_I(x_i), P_I(x_i) > 0 \\ -|\mathcal{S}|, P_I(x_i) = 0 \end{cases}}{|\mathcal{S}|} \tag{10}$$

3.1. Problem optimization

We optimize finding the solution in two steps. The first step is graph pruning, where the algorithm removes vertices and associated edges from the search space. The second step is introducing an algorithm that uses previously acquired knowledge about found solutions to speed up the evaluation of proposed solution.

3.1.1. Pruning the search space

Pruning the search space could be accomplished by repeatedly finding critical paths and applying all available security elements to nodes on critical paths until there are no new nodes found on critical paths. Nodes that are not included in a set of found on critical paths during the search, have no significance to optimization results. Therefore, they could be removed from the search space, together with associated edges. Pseudo code is shown in Figure 3.

Figure 4 shows a simple graph, used for an explanation of the pruning algorithm. A search for a critical adversary path, if there are no security elements implemented, would find path (A, B, E). In the first step, the pruning algorithm would place found nodes A, B and E to set $\mathcal{FN}$ and assign them to all available security elements to achieve maximum security for that path. The next search for a critical path would find (A, C, E) as the most critical. Node C is not present in set $\mathcal{FN}$ so the algorithm would place all assignable security elements to node C and add C to $\mathcal{FN}$. Therefore, the set of all found vertices in critical paths $\mathcal{FN}$ equals

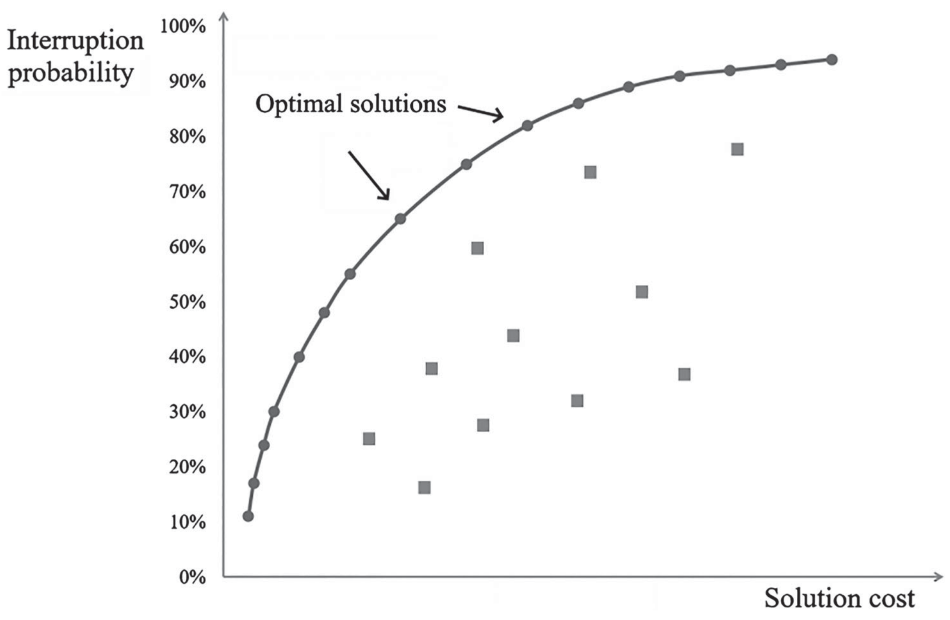


Figure 2. Pareto front in optimizing physical security system.

```
PruningSearchSpace algorithm

  $foundNodes \leftarrow \emptyset$

do
  $previouslyFoundNodes \leftarrow foundNodes$
  criticalPaths = FindCriticalPathsForAllEntryNodes()
  for each element v in criticalPaths
    ap(v) ← ae(v)
  add $v$ to $foundNodes$
while not $foundNodes$ is proper subset of $previouslyFoundNodes$

RemoveAllNodesFromSearchSpaceThatAreNotIn($foundNodes$)
```

Figure 3. Pruning search space algorithm.

{A, B, E, C}. Further search would find (A, B, E) or (A, C, E) as the new critical path. Since both possible result sets are proper subsets of found nodes set, there are no new nodes found and the algorithm can stop searching for new critical paths. The pruned search space is then created using vertices in $\mathcal{FN}$ set {A, B, E, C} and associated edges. Node D does not appear in any critical path, so it is safely removed from search space.

3.1.2. Domain experienced exploration (DEX2) method

For a facility that has five entry elements, fifteen positions to place security elements and 3 security elements that could be applied per each position, it would take $5 \times 2^{3 \times 15}$ or 1.76×10^{14} evaluations to calculate all possible solutions. Real facilities could have tens to hundreds of possible entries, thousands of positions to place security elements and choices for tens to hundreds of security elements. Therefore, brute-force calculation is not a viable option for this problem.

Searching for minimal cost of security system which achieves a maximum probability that adversary is stopped makes it a multiple, competing objectives problem. Simultaneous optimization of competing objectives, lowest system price and highest probability to stop adversary, can be achieved by combining those two values via *utility functions*, as a weighted sum, into a single scalar value that is optimized as a single objective [22]. Utility function is not well known prior to the

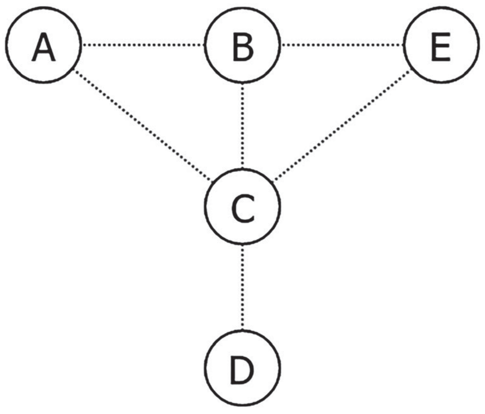


Figure 4. Simple graph for explanation of pruning algorithm.

optimization process so there are few drawbacks if the single objective optimization method is used for solving multi-objective optimization problem. The result of such method depends on weight coefficients so optimal solution could be unacceptable due to an inappropriate setting of coefficients and additional runs of the optimizer are required [23]. Another challenge is that in some problems small changes in coefficients could create significant change in the result [24].

The main drawback in using single-criteria optimization method for our problem is that it produces single result, compared to multi-objective evolutionary algorithms (MOEA) [25] which produce a set of Pareto optimal solutions, based on which decision maker can make informed decision. MOEA operate on a *generational population* that presents set of *individuals*, an encoded solution, which could be represented as the binary vector $x = (x_1, \ldots, x_b)$ where $b = \sum_{i=1}^{n} |ae(v_i)|$. Each bit represents if available security element is applied to associated construction element.

No matter which MOEA algorithm is chosen for finding the solution, for each individual there are many repeated changes of search space and consequently searches for critical paths. Applying security elements to construction elements changes vertices properties, affecting an adversary's critical path.

We introduce the algorithm that takes into consideration knowledge acquired during searches to avoid unnecessary calculation, as shown in Figure 5. The main concept is to check if the intersection of known solutions and proposed solution is an empty set. If proposed security elements do not have an impact on known critical path, then there is no need to calculate the critical path. If there is no knowledge about proposed security elements placement, then the algorithm will find the critical path, calculate solution quality and cost, and add it to the list of found solutions.

For the purpose of explanation of how domain experienced search algorithm operates, the sample graph is created, as shown at Figure 6, consisting of seven nodes {A, B, C, D, E, F, G}, connected with eleven edges, where the starting node is node A and the target node for adversary attack is node G. During the optimization process, different algorithms may propose individuals as problem solutions. An algorithm is creating tree-like structure of found solutions, organized in such a way that it's possible to decide if new a search is necessary or if existing solutions could be used, as shown in the next examples.

As shown at Figure 6, position 1, if there are no security elements placed, the optimal path for the adversary is {A, C, E, G}. For proposed individuals that place security elements at nodes B, D and F nodes only, the intersection with set {A, C, E, G} would be an empty set. That means that proposed security elements would not influence the change of the critical path. The proposed algorithm would not initiate an update of the search space and the calculation of the critical path would become unnecessary because known solution {A, C, E, G} already exists.

The evaluation of individual that consists of set {B, C, F} would take two steps. The intersection of the initial critical path {A, C, E, G} and individual {B, C, F} is set {C}. In the first step, an algorithm applies a security element at node C, updates the search space and searches for the critical path. The critical path in our example, when security element is applied at node C, is {A, B, D, E, G} as shown at position 2 of Figure 6. In the next step, the initial set with active nodes {A, C, E, G} is united with nodes that are on critical path {B, D} and

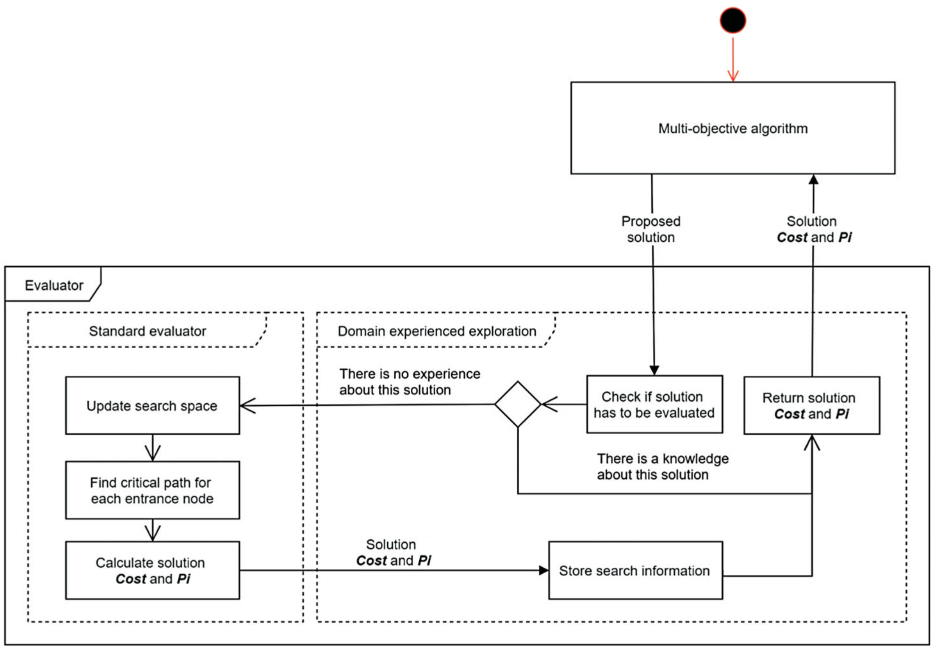

Multi-objective algorithm
Proposed solution
Solution Cost and PI
Evaluator
Standard evaluator
Domain experienced exploration
Update search space
Find critical path for each entrance node
Calculate solution Cost and PI
There is no experience about this solution
Check if solution has to be evaluated
Return solution Cost and PI
There is a knowledge about this solution
Solution Cost and PI
Store search information

Figure 5. Optimization flowchart using domain experienced exploration.

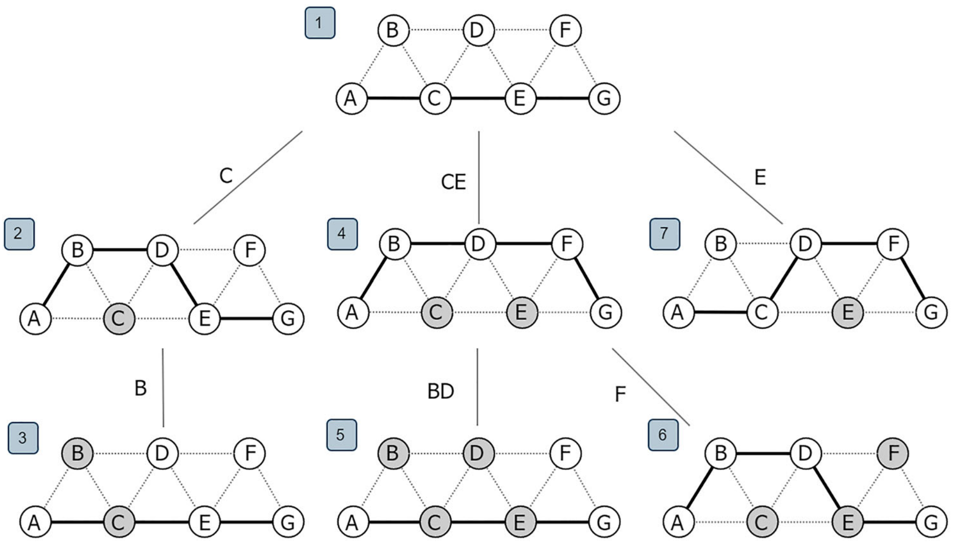


Figure 6. Explanation of domain experienced search algorithm.

now the algorithm calculates {A, B, C, D, E, G} ∩ {B, C, F} = {B, C}. The intersection is not an empty set, so the search space is updated by placing security elements at nodes {B, C} and new calculated critical path is {A, C, E, G}, as shown at position 3. Node F from initial individual has no influence at critical path {A, C, E, G} so further exploration is not necessary.

Based on this exploration, if individual {C, F} is evaluated, a new calculation of the critical path is avoided. In the first step {A, C, E, G} ∩ {C, F} equals {C} and there is already information about critical path {A, B, D, E, G} for the search space with active node C. Therefore, there is no need to calculate that critical path. Node F has no influence, so the previously calculated solution is returned, without the need to update the search space and find the critical path.

The second example is with individual {B, C, D, E} and in the first step {A, C, E, G} ∩ {B, C, D, E} equals {C, E}. There is not a known solution for {C, E} so security elements are applied and a new critical path {A, B, D, F, G}, as shown in position 4 of Figure 1, is found and stored as a new sub solution. In the next step, intersection {A, B, C, D, E, F, G} ∩ {B, C, D, E} equals {B, C, D, E}, and critical path {A, C, E, G}, shown in position 5, is found. The new intersection is the same, so this solution is returned as a result.

The individual in our third example is {C, E, F} and in the first step {A, C, E, G} ∩ {C, E, F} equals {C, E}. The algorithm finds existing sub solution, at position 4, with critical path {A, B, D, F, G}. In the next step, intersection {A, B, C, D, E, F, G} ∩ {C, E, F} equals {C, E, F}. There

```
Pseudo code DomainExperiencedExplorationEvaluator

Evaluate(individualVector)
for each $v_s \in S$
  if initialSolutions[$v_s$] is null
  .  searchSpace ← ApplySecurityElementsToSearchSpace(elements: ∅)
  .  initialSolutions[$v_s$] ← FindCriticalPath(searchSpace, $v_s$)

  possibleSolution ← initialSolutions[$v_s$]
  while ¬solutionFound do
  .  intersection ← individualVector ∩ possibleSolution.activeElementsSinceStart
  .  if ((intersection ≠ possibleSolution.activeElements) and (intersection ≠ ∅))
  .  .  if intersection ∈ possibleSolution.subsolutions
  .  .  .  possibleSolution ← possibleSolution.subsolutions[intersection]
  .  .  else
  .  .  .  searchSpace ← ApplySecurityElementsToSearchSpace(intersection.activePositions)
  .  .  .  newSolution = FindCriticalPath(searchSpace, $v_s$)
  .  .  .  possibleSolution.subsolutions[intersection] ← newSolution
  .  .  .  possibleSolution ← initialSolutions[$v_s$]
  .  else
  .  .  solutionFound ← true

  return possibleSolution.criticalPath
```

Figure 7. Domain experienced exploration algorithm pseudo-code.

is not an existing sub solution, so the algorithm applies security elements at {C, E, F} and searches for the critical path. The found solution {A, B, D, E, G}, shown in position 6, is added as a sub solution of {A, B, D, F, G}, position 4.

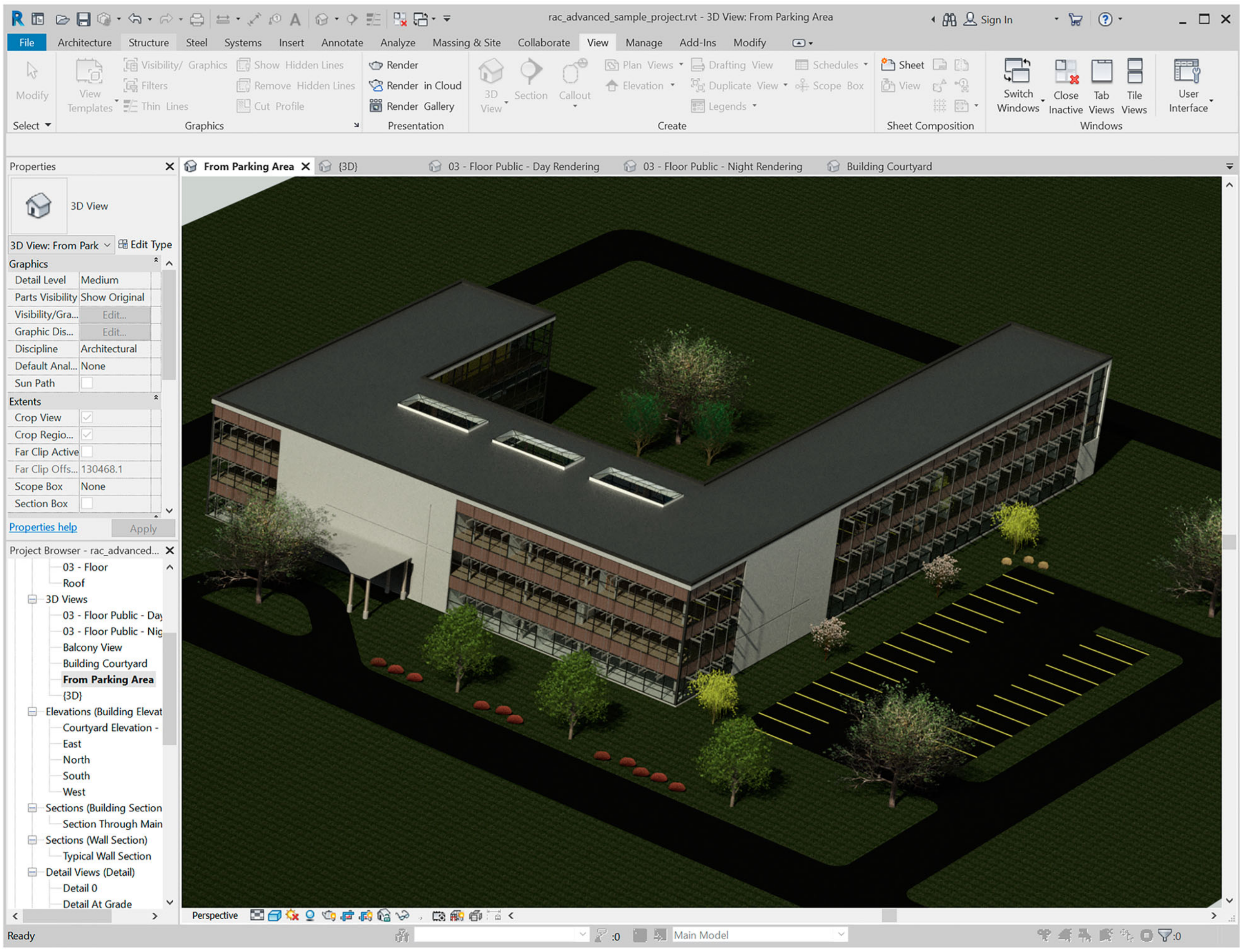
The last example is with individual {B, E}. In the first step {A, C, E, G} ∩ {B, E} = {E} and the new critical path is {A, C, D, F, G} (position 7). Node B is not on the critical path, so exploration is not continued.

Algorithm details are presented in Figure 7.

4. Experimental setup

4.1. Test dataset and test environment

A multi-objective search for an optimal solution was performed using the HeuristicLab framework for evolutionary algorithms that was developed by members of



3D View: From Park | Edit Type

Graphics

Detail Level	Medium
Parts Visibility	Show Original
Visibility/Gra...	Edit...
Graphic Dis...	Edit...
Discipline	Architectural
Default Anal...	None
Sun Path	

Extents

Crop View	✓
Crop Regio...	✓
Far Clip Active	
Far Clip Offs...	130468.1
Scope Box	None
Section Box	

Figure 8. Autodesk Revit “rac_advanced_sample” building.

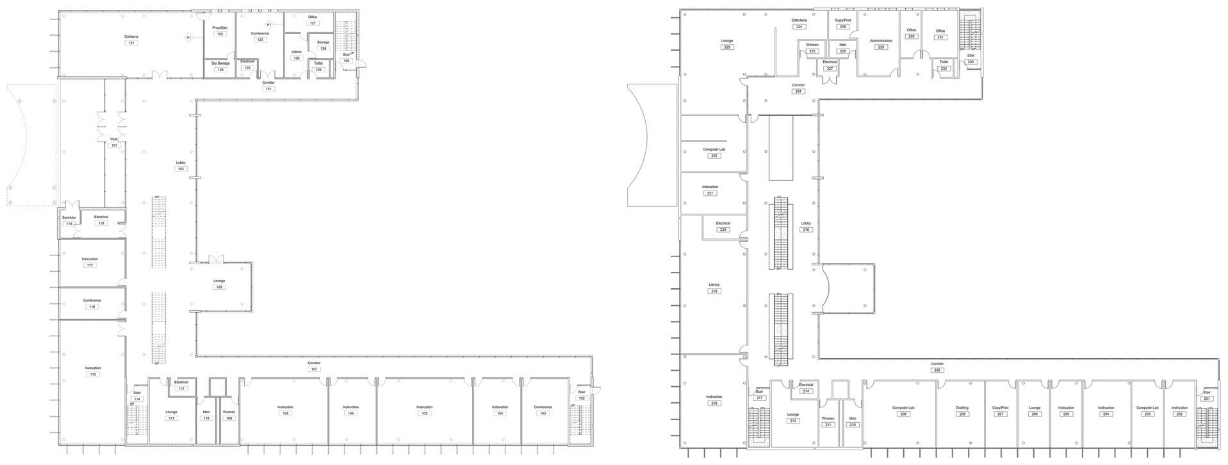


Figure 9. The entry level and first level floor plans of RAC_advanced_sample.

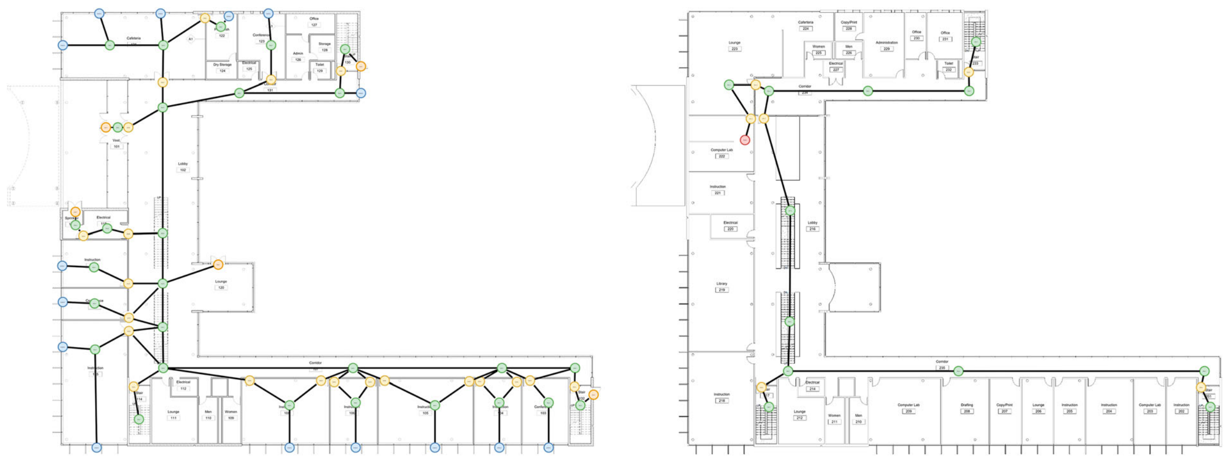


Figure 10. Test facility A, represented as graph.

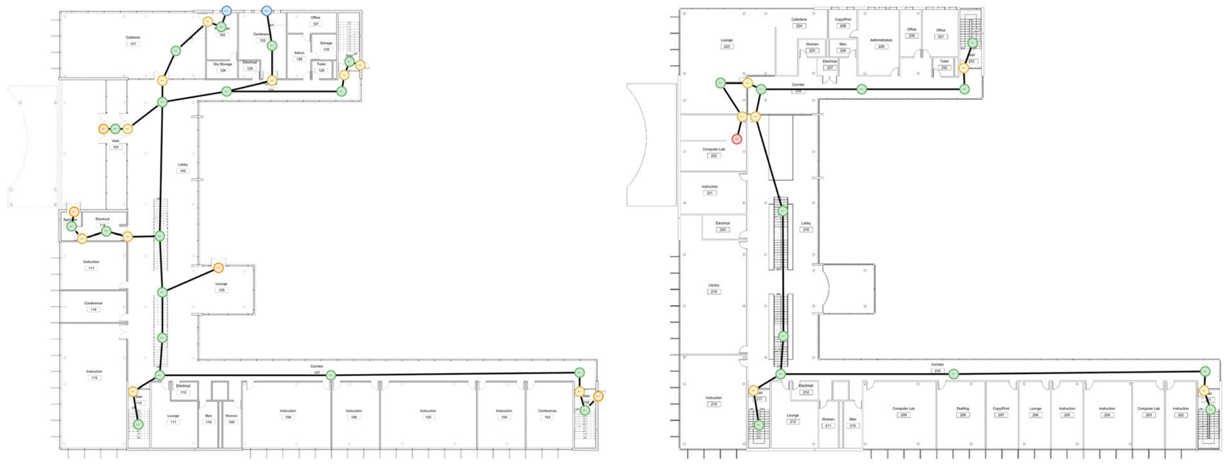


Figure 11. Test facility B, represented as graph.

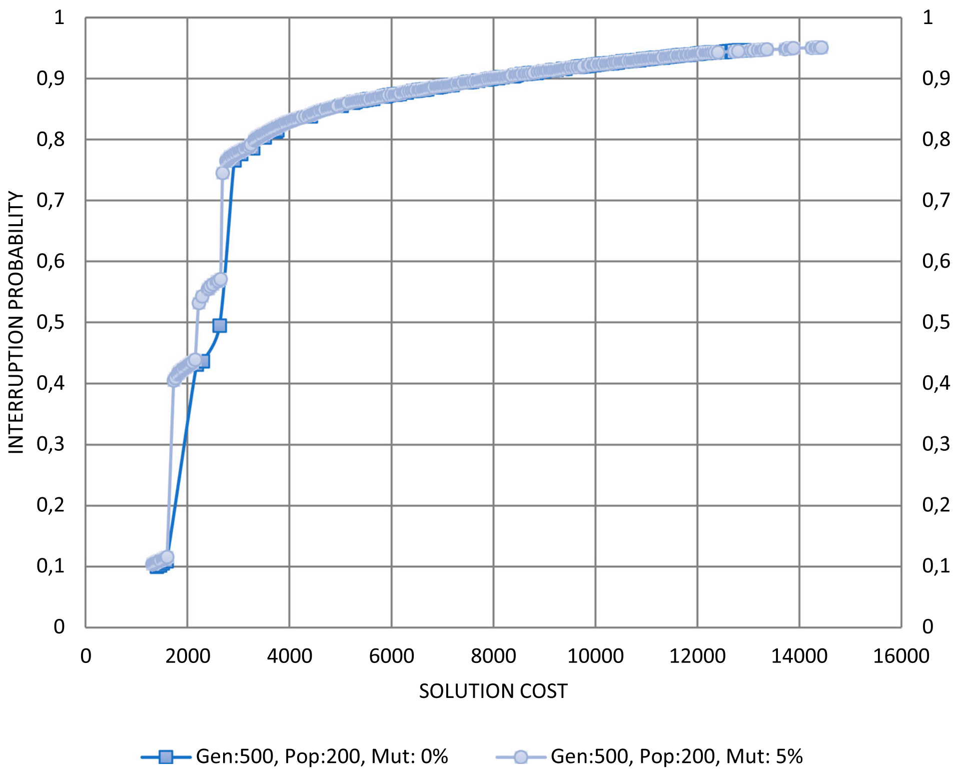


Figure 12. Results of Facility A optimization.

the Heuristic and Evolutionary Algorithms Laboratory (HEAL) since 2002 [26]. Multiple objective optimization was performed using a computationally fast elitist evolutionary algorithm based on a non-dominated sorting approach NSGA-II (Non-dominated Sorting Genetic Algorithm II) [27]. It finds the spread of solutions near the true Pareto-optimal front. Population size, number of evaluations, mutation and crossover probabilities, presented in experiments results, are selected after numerous experiments with different parameters of genetic algorithms. Lower numbers of population size and the number of evaluations generated lower quality results, while increasing them didn't contribute to better results.

Evaluation algorithm was written in. Net Framework, using C# programming language syntax.

Facility samples used to run tests were made using "RAC_Advanced_sample", available as Autodesk Revit® 2020 distribution. Building has three floors – entry level, first and second floor, as shown at Figure 8.

Entry level and first floor were used to create test models (Figure 9). Computer room on first floor (room no. 222) has been selected as target position.

Two test models were created to evaluate a dependence of domain experienced exploration evaluator performance to the number of facility elements and dependence on number of entry elements. The first test model represents a standard commercial building. The building has glass elements at entry levels that an adversary can use for entrance into the building. Glass elements at first floor were not used as possible entrance elements in this experiment. The second test model represents the same building as the high security facility. Glass elements are replaced with walls and only two windows are available as entry points.

The first test model (Facility A), as shown in Figure 10, is represented as a graph, consisting of 85 nodes, where 20 of them are entry nodes. The nodes are denoted with different colours that represents element type. Yellow nodes represent doors, green nodes represent rooms and blue represent windows. Rooms and doors that are not represented as nodes are not important as they would be pruned in first step of the optimization process.

Second test model (Facility B), is shown in Figure 11 and, represented as a graph, consists of 50 nodes, where 7 of them are entry nodes.

Table 1. Experimental results for facility A.

Search no.	Running sum of evaluations	Running sum of DEX2 evaluations	Avoided evaluations	New solutions found	Improved solutions
Algorithm:	NSGA-II, Population size: 500, Generations: 200, Mutation probability: 0%, Crossover probability: 90%				
1	2.010.000	1.087.000	46%	435	–
2	4.020.000	2.197.108	45%	295	124
3	6.030.000	3.506.118	42%	191	116
4	8.040.000	4.623.331	42%	149	181
5	10.050.000	5.755.194	43%	68	90
	NSGA-II, Population size: 500, Generations: 200, Mutation probability: 5%, Crossover probability: 90%				
1	2.010.000	1.524.477	24%	451	–
2	4.020.000	3.171.812	21%	276	81
3	6.030.000	4.595.692	24%	213	191
4	8.040.000	6.099.798	24%	137	118
5	10.050.000	7.355.339	27%	93	280

5. Experimental results

Figure 12 shows two sets of solutions proposed for Facility A, presented as Pareto fronts, found using

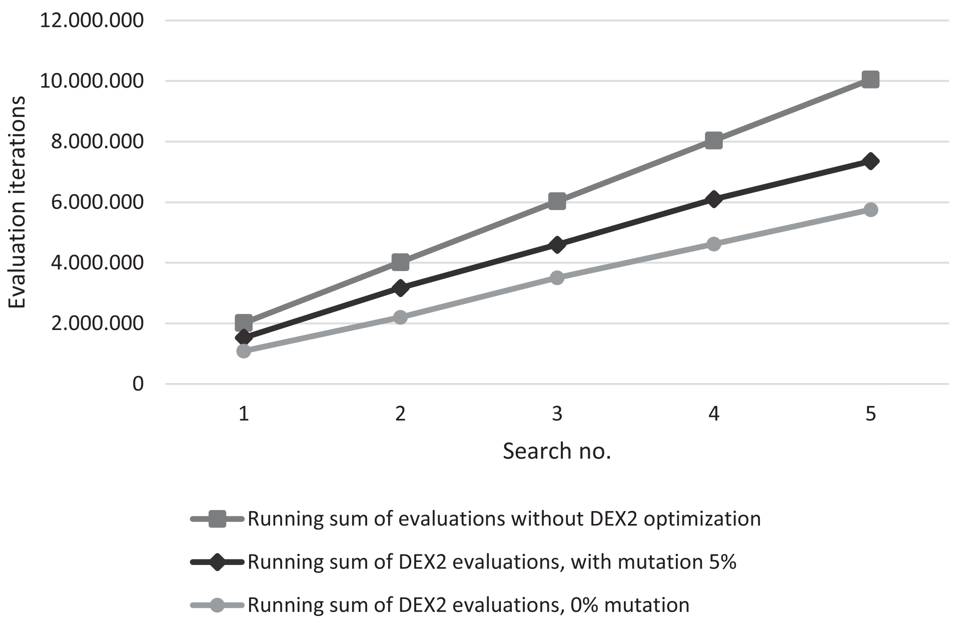


Figure 13. Comparison of iterations with and without DEX2 – Facility A.

Table 2. Experimental results for facility B.

Search no.	Running sum of evaluations	Running sum of DEX2 evaluations	Avoided evaluations	New solutions found	Improved solutions
Algorithm:	NSGA-II, Population size: 200, Generations: 200, Mutation probability: 0%, Crossover probability: 90%				
1	281.400	206.839	26%	189	–
2	562.800	340.533	39%	135	25
3	844.200	523.768	38%	85	55
4	1.125.600	685.735	39%	66	29
5	1.407.000	886.320	37%	67	53
	NSGA-II, Population size: 300, Generations: 300, Mutation probability: 5%, Crossover probability: 90%				
1	632.100	323.909	49%	247	–
2	1.264.200	589.623	53%	169	36
3	1.896.300	918.673	52%	122	13
4	2.528.400	1.242	51%	70	60
5	3.160.500	1.520	52%	54	79

NSGA-II with two algorithm setups. Both setups run genetic algorithm with 500 generations, each having 200 individuals. The first setup does not use the mutation method, and the second setup mutates 5% of parent individuals when creating the new generation. As the graph in Figure 12 shows, the mutation method helps in finding more diverse solutions.

The first experiment results, presented in Table 1, compare the number of evaluations with and without a knowledge-based evaluator for Facility A. The NSGA-II algorithm is run in five sequential runs and the domain exploration algorithm keeps exploration knowledge for each run. Solution results found after each run are compared to all previous results to calculate the count of new and improved results. Table 1 shows results for two NSGA-II setups, and difference between them is that column "Running sum of evaluations" values are the running sums evaluation when DEX2 algorithm is not used. In this experiment, each NSGA-II run executes 2.010.000 evaluations. Column "Running sum of DEX2 evaluations" presents the running sum of number of evaluations when domain experienced exploration algorithm is applied to evaluation process. Comparisons of running sums of evaluations, with and without DEX2 applied, show that usage of domain experienced exploration algorithm decreases the number of iterations, as shown in column "Avoided evaluations" and graphically presented in Figure 13. Mutation probability decreases algorithm optimization effectiveness as more mutated individuals require more significant branching in the search tree. Column "New solutions found" presents count of new solutions found, compared to all previous NSGA-II runs. Column "Improved solutions" is the number of newly found solutions that offer higher probability that an adversary will be stopped, compared to previously found solutions with the same budget.

Figure 14 shows two sets of solutions proposed for Facility B, presented as Pareto fronts, found using NSGA-II with two algorithm setups. The first algorithm setup has 200 generations, 200 individuals per generation and doesn't use the mutation method, while second setup has population of 300 individuals, 300 generations and 5% mutation. A bigger population size and greater number of generations in the second setup

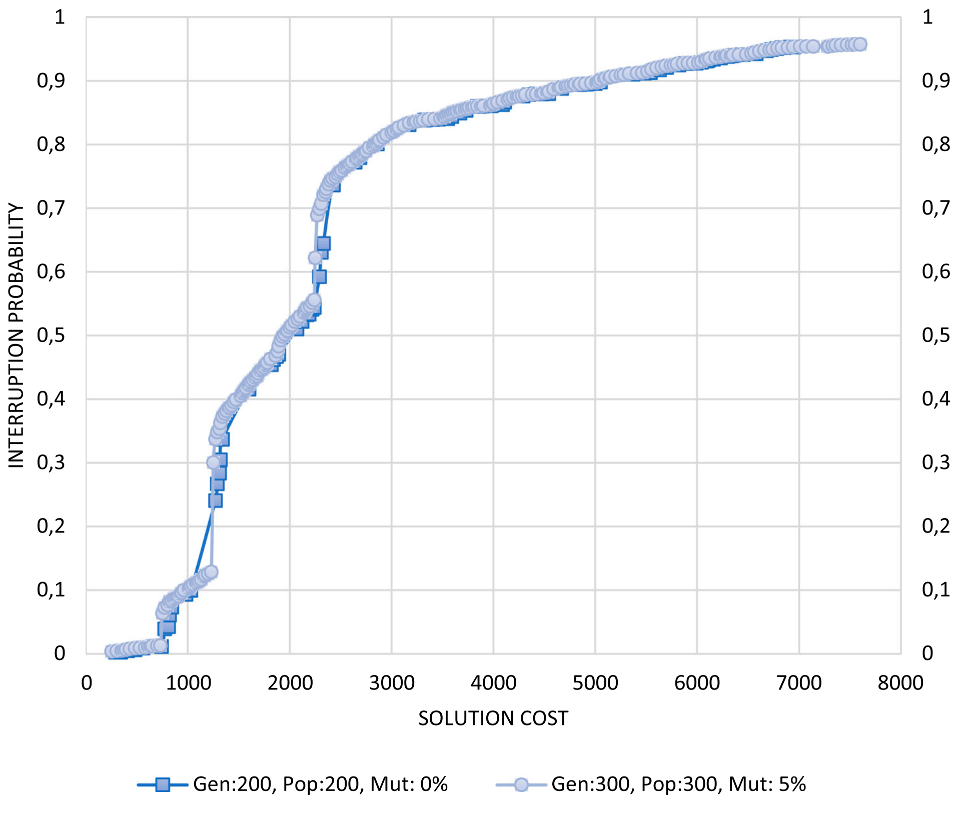


Figure 14. Results of Facility B optimization.

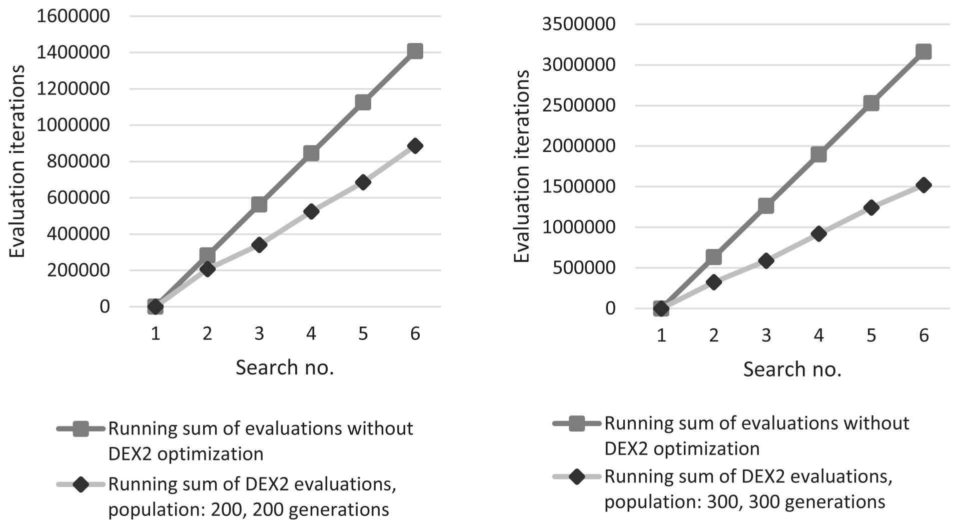


Figure 15. Comparison of iterations with and without DEX2 – Facility B.

find more dominant solutions compared to the first setup, as seen in Figure 14.

Experiment results for facility B are presented in Table 2. The goal of this experiment is to explore the influence of graph branching and the number of generations to DEX2 algorithm effectiveness. Results show that, compared to the previous experiment, larger number of entry nodes and graph branching increases algorithm effectiveness. Also, as Figure 15 shows that, a higher number generations produce a greater number of similar solutions that causes the algorithm to have higher effectiveness.

6. Conclusion

The objective analysis and optimization of physical protection systems require mathematical model and multi-objective search algorithm. In real-world scenarios, facilities that are being protected are comprised of a large number of construction elements and a variety of available security elements. In this paper we presented a method that reduces the number of evaluation searches, by initial pruning of search space and by using experience from previous domain exploration. For each proposed solution the algorithm will check if there is existing knowledge that helps to avoid any unnecessary search for critical paths and calculation of cost and solution quality. The benefit of applying this algorithm is that reducing the number of evaluations shortens the time of the optimization. It is not an optimal method if the graph model representing the facility layout has the form of a grid, in which case the new critical paths are predominantly found for each proposed solution and the algorithm cannot use previously found results.

Our algorithm was empirically evaluated on two facilities that represent real-world projects, having different problem sizes and facility layouts. We used NSGA-II as an evolutionary multi-objective algorithm to propose solutions and to create Pareto front. Experiments showed that our method improves evaluation by pruning the search space and avoiding unnecessary searches, where some results showed significant reduction in searches. The advantage of our algorithm concept is that accumulated domain exploration knowledge can be used with different multi-objective algorithms, configured with various parameters, run at same problem, avoiding unnecessary searches. Example is running the genetic algorithm with different parameters.

Proposed evaluation algorithm could be used in other problem domains where search space, and consequently critical path, changes dynamically based on proposed solution.

Disclosure statement

No potential conflict of interest was reported by the authors.

ORCID

Marin Golub http://orcid.org/0000-0002-8042-7076

References

[1] Garcia ML. The design and evaluation of physical protection systems. 2nd ed. Amsterdam: Elsevier/Butterworth-Heinemann; 2008.

[2] Matko V, Brezovec B. Improved data center energy efficiency and availability with multilayer node event processing. Energies. 2018;11(9):1–17.

[3] Jia M, Srinivasan RS, Raheem AA. From occupancy to occupant behavior: an analytical survey of data acquisition technologies, modeling methodologies and simulation coupling mechanisms for building energy efficiency. Renew Sustain Energy Rev. 2017;68:525–540.

[4] Brezovec B, Matko V. Software and equipment for remote testing of sensors. Sensors. 2007;7(7):1306–1316.

[5] Doyon LR. Stochastic modeling of facility security-systems for analytical solutions. Comput Ind Eng. Jan. 1981;5(2):127–138.

[6] Garcia ML. Vulnerability assessment of physical protection systems. Amsterdam: Elsevier Butterworth-Heinemann; 2006.

[7] Jang S-S, Kwan S-W, Yoo H-S, et al. Development of a vulnerability assessment code for a physical protection system: systematic analysis of physical protection effectiveness (SAPE). Nucl Eng Technol. 2009 Jun;41(5):747–752.

[8] Porter S, Tan T, Tan T, et al. Breaking into BIM: performing static and dynamic security analysis with the aid of BIM. Autom Constr 2014 Apr;40:84–95.

[9] Zou Y, Kiviniemi A, Jones SW. A review of risk management through BIM and BIM-related technologies. Saf Sci. 2017 Aug;97:88–98.

[10] Čakija D, Ban Ž. Modeling facility protection for numerical vulnerability assessment. 2011 Proc. 34th Int. Conv. MIPRO; 2011.

[11] Edelkamp S, Schrödl S. Heuristic search: theory and applications. Amsterdam: Morgan Kaufmann; 2012.

[12] Hart P, Nilsson N, Bertram R. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybern. 1968;SSC-4:101–107.

[13] Zou B, Yang M, Zhang Y, et al. Evaluation of vulnerable path: using heuristic path-finding algorithm in physical protection system of nuclear power plant. Int J Crit Infrastruct Prot. 2018 Dec;23:90–99.

[14] Rios LHO, Chaimowicz L. PNBA*: a parallel bidirectional heuristic search algorithm. p. 12.

[15] Holte RC, Felner A, Sharon G, et al. Bidirectional search that is guaranteed to meet in the middle. AAAI16 Proc. Thirtieth AAAI Conf. Artif. Intell.; 2016. p. 3411–3417.

[16] Zou B, Yang M, Guo J, et al. A heuristic approach for the evaluation of physical protection system effectiveness. Ann Nucl Energy. 2017 Jul;105:302–310.

[17] Gargano ML. Evolving efficient security systems. 2003 May; p. 10.

[18] Flammini F, Gaglione A, Mazzocca N, et al. Optimisation of security system design by quantitative risk assessment and genetic algorithms. Int J Risk Assess Manag. 2011;15(2/3):205.

[19] Flammini F, Gentile U, Marrone S, et al. A petri net pattern-oriented approach for the design of physical protection systemsComputer Safety, Reliability, and Security: 33rd International Conference, SAFECOMP 2014 Vol. 8666. Florence, Italy: Springer; 2014. p. 230–245.

[20] Brown NJK, Jones KA, Nozick LK, et al. Multi-layered security investment optimization using a simulation embedded within a genetic algorithm. In: 2015 Winter Simulation Conference (WSC), Huntington Beach, CA; 2015. p. 2424–2435.

[21] Brown N, Jones K, Bandlow A, et al. A stochastic programming approach to the Design optimization of Layered physical protection systems. Presented at the Hawaii International Conference on system Sciences; 2017.

[22] Fonseca CM, Fleming PJ. Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. p. 8.

[23] Coello C, Romero C. Evolutionary algorithms and multiple objective optimization. In: Ehrgott M, Gandibleux X, editors. Multiple criteria optimization: state of the art annotated bibliographic surveys. Boston, Dordrecht, London: Kluwer Academic Publishers; 2002. p. 277–331.

[24] Caballero R, Rey L, Ruiz F, et al. An algorithmic package for the resolution and analysis of convex multiple objective problems. In: Fandel G, Gal T, editors. Multiple criteria decision making, Proceedings of the 12th International Conference. Hagen, Germany: Springer; 1997. p. 275–284.

[25] Coello CAC, Lamont GB, Veldhuizen DAV. Evolutionary algorithms for solving multi-objective problems second edition. Boston (MA): Springer; 2007.

[26] Wagner S, Kronberger G, Beham A, et al. Architecture and design of the HeuristicLab optimization environment. In: Klempous R, Nikodem J, Jacak W, et al., editors. Advanced methods and applications in computational intelligence Vol. 6. Heidelberg: Springer International Publishing; 2014. p. 197–261.

[27] Deb K, Pratap A, Agarwal S, et al. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput. 2002 Apr;6(2):182–197.