2

Optimizing a Physical Security Configuration Using a Highly Detailed Simulation Model

Tom Marechal
Eindhoven University of Technology

Alice E. Smith
Auburn University

Volkan Ustun
Auburn University

Jeffrey S. Smith
Auburn University

Erjen Lefeber
Eindhoven University of Technology

2.1 Introduction

When a building or a set of buildings needs a physical security system to prevent intrusions, it can be hard to find a configuration that is cost effective but also performs the security task adequately. Especially, when both human guards and security cameras are used it is difficult to find a near-optimal configuration because of the number of potential designs. How many guards are needed and where should each walk? How many cameras are needed, where should each be located and what direction should each point at? These questions are usually answered by using intuition and common sense. The problem when using intuition and common sense is that it is possible to over-secure or under-secure a certain part of the building. Over-securing results in costs that are higher than necessary while under-securing might lead to intruders breaking in successfully, potentially with catastrophic consequences.

In this chapter, a highly detailed simulation model has been used to find a superior security configuration to secure a building. The building has to be protected against intruders who want to steal items. One could, for example, envision a sensitive area where important paperwork is to be protected from being stolen by an external enemy. Or, possibly a weapon or fuel storage depot that has to be secured from enemy intrusions. In this chapter, a stochastic simulation model is coupled with a heuristic optimizer to find a security configuration that is as cost effective as possible while keeping the probability

that an intruder breaks in and steals one or more items near zero. The simulation is used to probabilistically evaluate security configurations while the heuristic is used to put forth candidate security configuration designs.

In the next section, a literature review is given. A simulation model of an actual building with guards, security cameras and intruders has been made. The objective functions and the constraints are given in the Section 2.3. In Section 2.4, more details on the simulation are given. The binary problem representation and the newly developed optimization method (hybrid genetic ant optimization, HGAO) are explained in detail in the Section 2.5. In the Section 2.6, the outcome of the HGAO heuristic on two security optimization problems is given. Conclusions are drawn in Section 2.7. Finally, recommendations for further research are given in Section 2.8.

2.2 Literature Review

In this section, literature with respect to the optimization problem and literature with respect to simulation are given. After that a review on heuristics that are strongly related to the heuristic used are provided.

2.2.1 Optimization Problem

Physical security systems to prevent intrusions have been studied in the literature before. However, usually the studies focus on only a small part of the security system, for example, on the working of a single movement sensor or a single laser sensor. Only a very small number of studies focus on more comprehensive security systems. Olario and Nickerson [9] provide an in-depth study on several sensor configurations to protect one single valuable item. In Cavalier et al. [3] several optimization approaches are performed to find a good security sensor configuration for a greatly simplified model. In that study the probability of detection is a decreasing function of distance between intruder and sensor. It is suggested that empirical research methods should be done to investigate in what way distances from different sensors influence detection probabilities. In Jones et al. [5] arcs and nodes are used to evaluate a security system with Markov processes. Success probabilities for different arcs of the intruder path are estimated in advance. The main focus is to find the way at which an intruder has the highest probability to do harm, using low-detailed information. In Jordon et al. [6] performances of complete security systems are tested. In that study the testing is done both by applying low detailed analytical estimates and by applying simulation using human participants. For the latter purpose a real-time connection to a virtual reality model has been used to incorporate real human behavior.

To the best of our knowledge, the only studies on using a highly detailed simulation to evaluate physical security systems are Hamel et al. [4] and Smith et al. [10]. In Hamel et al. [4] an effective sensor network configuration to detect nuclear, biological or chemical attacks is determined. Detailed information such as location and specifications of buildings and different meteorological situations are used. Note that the type of problem in Hamel et al. [4], detecting a nuclear, biological or chemical attack, is quite different from this study. In Hamel et al. [4], mitigation of the threat is an aim over just detection while this chapter will focus on detection. In Smith et al. [10] a simulation using simple agents to represent intruders and guards is introduced. Many realistic features such as doors, windows, locks and security cameras have been implemented in the simulation. The simulation is used to evaluate predefined security configurations.

No literature describes using a highly detailed simulation to *optimize* the security system. In this chapter the simulation of Smith et al. [10] is used as a base. Some additional features have been added and a graphical engine has been built in. Some minor modifications have been made and some additional features have been added to put even more detail into the simulation. Furthermore, the line-of-sight module that evaluates whether an intruder can be seen by a guard or not has been replaced by a faster line-of-sight module, as described in Ustan et al. [11].

2.2.2 Optimization Heuristics

Because of the large search space of the decision variables (camera numbers and locations, guard numbers and paths) and the relatively lengthy simulation needed to evaluate each proposed security configuration, the problem is posed as a binary (combinatorial) problem and solved with a heuristic. The optimization is performed with a new hybrid heuristic, combining aspects of a binary genetic algorithm and of a binary ant colony optimization. Binary genetic algorithms have been used before in various research fields, two of the many examples are Ananda et al. [1] and Yeo and Lu [12]. Binary ant colony optimization has been used before too, for example, Kong and Tian [7] and Nahas and Nourelfath [8]. In Bu et al. [2] a means to turn continuous problems into binary problems has been developed, so continuous problems can be solved as binary problems too, expanding the use of the binary optimization method described in this study.

2.3 Problem Formulation

The simulation, the objective function and the constraints are devised in such a way that several different optimization problems can be solved.

2.3.1 Objective Function

The objective is to minimize the total costs which are all viewed as periodic (annual in this case). Total costs consist of guard salary, yearly camera value deduction, damage to the building caused by an intruder and the value of any stolen items.

$$\begin{matrix}\text{Total}\\ \text{costs}\end{matrix} = \begin{matrix}\text{Guard}\\ \text{salary}\end{matrix} + \begin{matrix}\text{Camera value}\\ \text{deduction}\end{matrix} + \begin{matrix}\text{Damage to}\\ \text{building}\end{matrix} + \begin{matrix}\text{Value of}\\ \text{stolen items}\end{matrix} \tag{2.1}$$

Guard salary depends on the number of guards—each additional guard incurs the same additional annual salary. Camera value deduction depends on the number of cameras in the same manner. That is, both guard costs and camera costs are linear. Damage to buildings can consist of the breaking of doors, windows or locks when an intruder attempts to break in. Furthermore, an intruder can also induce damage at other points in the building, for example when s/he is collecting an item to steal (if for instance the item was stored in a locked display case that has to be broken).

Items are only accounted to be stolen if the intruder exits the building with the item. If the intruder gets caught before leaving the building or the area of buildings, the item itself does not contribute to the total costs. Damage that has been done to the building before the moment that an intruder gets caught does contribute to total costs.

The camera value deduction accounts for camera depreciation and possible maintenance costs, as well as power to operate it. In practice the camera value deduction per year is negligible compared to the guard salary; however, there was no reason to leave it out of the objective function. One might at a first glance think that if camera costs are negligible, the numbers of cameras in the optimal case will be extremely large. In Section 2.4, it is explained why this is not the case.

2.3.2 Constraints

While costs are the objective, the security requirement is handled as a constraint. This can be varied to suit the scenario considered. If for example a storage depot for explosives is to be secured, this constraint must be set to a very high probability of intruder detection and apprehension. In this chapter two constraint functions have been created. The minimum required fraction of intruders that get caught by a guard can be set as a constraint (the apprehension constraint). Furthermore, the minimum

required fraction of intruders that will be detected at some point can be set (the detection constraint). An intruder is detected either if a guard directly sees the intruder, or if the guard sees the intruder on a surveillance monitor.

$$\text{Fraction of intruders that get caught} \geq h_1 \tag{2.2}$$

$$\text{Fraction of detected intruders} \geq h_2 \tag{2.3}$$

2.3.3 Possible Optimization Problems

In the objective function (Equation 2.1), any of the four factors that contribute to the cost can be turned off if desired. The objective function has been described in this way to enable a variety of security problems to be tackled. If none of the factors are turned off, the objective will be to find the security configuration with lowest yearly costs. In that case some intruders might break into the building, causing damage to doors or windows. It might even be the case that some intruders steal one or more items without being caught. If one would not want intruders to ever succeed in stealing an item, h_1 in Equation 2.2 should be set to Equation 2.1.

One could also choose to turn off the third and fourth factors in the objective function (Equation 2.1) to find the security configuration that is as cheap as possible, ensuring that no more than a fraction of $1\text{-}h_1$ intruders break in and steal an item. One could also choose to only turn off the fourth factor, for example, if the value of stolen items cannot be expressed in terms of money.

The simulation and optimization are created in such a way that it is also possible to fix the cameras used (or not to use any cameras) and optimize the guards for that camera configuration. It is also possible to consider monitored cameras only without guard paths.

The optimization problems described above are just a few of the possibilities. Because of the fact that the simulation and the optimization are strongly decoupled and because of the fact that many factors can either be turned on or off, the simulation and optimization can be used to solve a variety of optimization problems.

2.4 The Simulation

Simulation is the tool that is used to verify the security designs in this chapter. When a promising design (a security configuration) has been identified, it is fed into the simulation. Based on this input, the simulation produces an output, i.e. the expected total costs, see Equation 2.1. Due to stochastic behavior in the simulation, the simulation is run numerous times for each design.

Although the simulation is very detailed, many assumptions have been made to simulate and run intruder scenarios and to test security configurations. It is hard to devise realistic scenarios that cover the range of possibilities. One should for instance answer the question where intruders would break in, how often they would break in and how they would behave once they were inside. One should also make several assumptions regarding guard behavior. However, to assist with this task the real-time graphical visualization of the simulation can immediately show if anything unrealistic is happening. The currently implemented aspects of the simulation as well as the assumptions made are described in Sections 2.4.1 through 2.4.6.

Section 2.4.1 describes the building that has been used in the simulation. In Section 2.4.2 features of the simulation are given. Section 2.4.3 describes assumptions that have been made in the simulation. In Section 2.4.4 the intruder scenarios are defined. Section 2.4.5 describes the possible guard patrol paths and Section 2.4.6 describes the camera features.

2.4.1 The Building

The purpose of this study is to find a good security configuration for a certain building or set of buildings. The building that was chosen for study is a three floor actual newly constructed building. The

building has one elevator and two stairways. All together, there are just over 100 rooms. The building has nine doors to the outside and 99 windows to the outside. A few simplifications have been made: all three floors are considered identical (except for doors to the outside which only exist at the ground floor), doors are assumed to be exactly as thick as walls, and both stairways and the elevator are modeled as rectangular hollow shafts. (Exact building data are not included because of their volume, but are available from the authors.)

2.4.2 Simulation Features

The goal of the simulation is to see how well a certain security configuration behaves for a depicted set of intruder scenarios. In the visualization one can see the intruders and the guards moving around and one can see the cameras. One can also see the areas that guards and cameras can view at each moment in time. Although the graphical representation is a 2-dimensional visualization, the entire model is 3-dimensional. The goal of the real-time graphical representation is to verify that the simulation model works as intended.

Intruders and guards walk a predefined path with a predefined speed at each arc of the path. Behavior of intruders is not determined on the fly, so an intruder will not alter behavior if s/he sees a camera or a guard. "Barriers" are ceilings, floors, walls, doors and windows. Doors and windows are "portals". Doors and windows can be opened, closed or even locked. Intruders can break locks of doors or windows. An agent (guard or intruder) can go to another floor by using the elevator or the stairs.

A line-of-sight algorithm [11] calculates whether a guard can see an intruder or not. In summary, the line-of-sight algorithm first determines a field of vision for each entity or camera, based on its viewing direction, on its maximum viewing distance (a preset parameter) and on the width of its viewing angle (also a preset parameter). After this, the algorithm checks if there are any points of other entities within this field of vision. (Each entity is represented by six points, one on the head, one on each arm, one on each foot and one on the stomach.) If there are any entity points within the field of vision, a check is done to see whether these points are actually visible, or if barriers (such as walls, ceilings or nontransparent doors) obstruct the view. The view of cameras is projected on surveillance monitors located in central bank. Whether the intruder can be seen by the camera is also determined by the line-of-sight algorithm. An intruder who is seen by a camera can only be detected if a guard is actually looking at the surveillance monitors. Only a fixed number of camera views can be projected on the surveillance monitors at the same time, if there are more cameras a switch is made every few time units. Figure 2.1 shows a typical screen shot of the simulation. In this figure, solid black lines are portals and the cameras, guards and intruder are all shown with their field of vision for that moment in time.

2.4.3 Simulation Assumptions

While the simulation used is quite detailed and flexible, certain assumptions regarding the facility and agent behavior must be made. These assumptions are common sense for the most part, however, some security scenarios may need different sets of assumptions:

- Initially all outside doors and windows are locked, some of the inner doors are locked too.
- A guard can go through every door quickly (that is, has a key for locked doors).
- An intruder can go through any door or window, but if the door or window is locked it will take considerably more time.
- Intruders can induce costs at any point in the building, defeating a locked door or window will induce costs, but an intruder can also cause damage at other places.
- A guard does not realize that an intrusion has occurred if s/he detects that the state of a door or window has been changed.
- An intruder walks slower if carrying a heavy item that has been stolen.

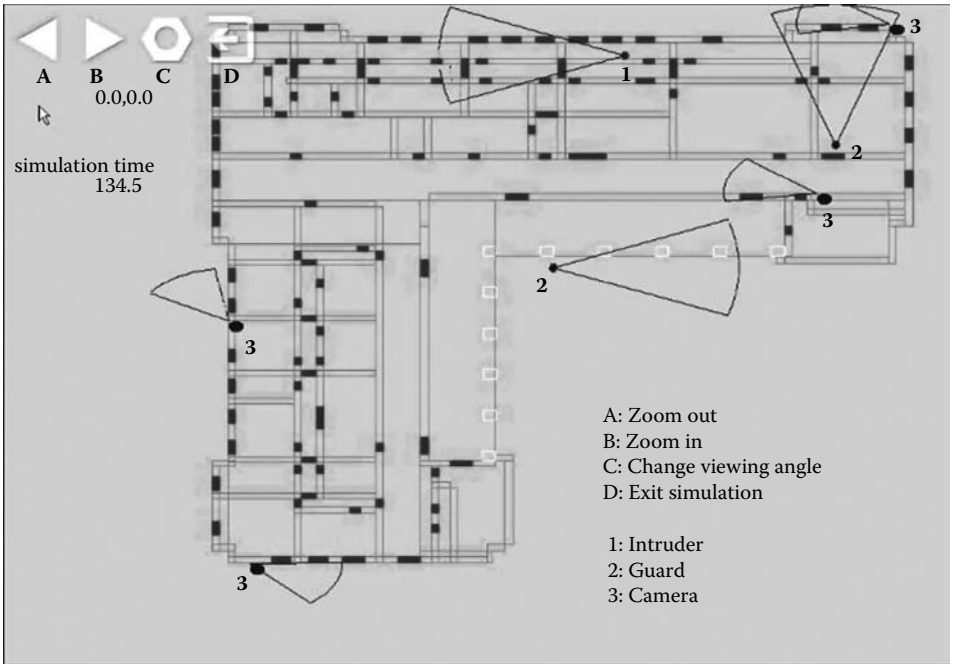

FIGURE 2.1 Screen shot of the 2-dimensional graphic visualization of the simulation.

- There is never more than one intruder at a given time.
- An intruder is detected if a guard sees her/him. A guard sees the intruder with a certain probability if the intruder is in the area the guard is physically capable to see.
- If an intruder is in sight of a camera, the intruder is detected with a certain probability if that camera is projected on the surveillance monitor and a guard is watching the surveillance monitor.
- No attempt has been made to simulate guards that actually chase the intruder. A calculation is done to determine the time an intruder has left after detection before s/he is caught. The amount of time depends on the distance between guards and the intruder, on the guard's running speed and on the floor the guards and intruder are. If the intruder is detected via camera, the guards first need a few seconds to determine where to go to. This delay could be caused by communication between guards on the intruder position for example.
- An item is only considered stolen if the intruder exits the building with it.

2.4.4 Intruder Scenarios

To test the optimization heuristic, certain intruder scenarios, candidate guard patrol paths and candidate camera options and positions have been defined. All of these settings can easily be changed to test for different situations. For this study, 40 intruder scenarios have been defined with the following assumptions:

- The number of times intruders try to break in per year is fixed.
- The largest proportion of the costs caused by an intruder consists of the value of stolen item(s).
- An intruder will only break in via a door or window on the first floor. However, some intruders go to the second or third floor via the stairs or via the elevator after breaking in on the ground floor.
- Intrusion scenarios where intruders go to the second or third floor of the building are less likely to happen than intrusion scenarios where intruders stay at the first floor.
- Some intruders only steal one item, some intruders steel multiple items during one intrusion, some intruders go in and out of the building several times each time stealing one or more items.

- Intruders do not always exit the building at the same place where they entered, but in most cases they do.
- In general, intruders move fast, except when they are carrying a heavy item.

2.4.5 Guard Patrol Paths

Seventeen candidate guard patrol paths have been defined. The following assumptions have been made:

- Some guard paths are outside the building, some are inside the building.
- Guard salary is $31 per hour per guard.
- The building needs to be guarded for 60 hours a week, 52 weeks a year.

2.4.6 Camera Features

Forty-one candidate camera positions have been defined. The following assumptions have been made:

- Some of the cameras are outside the building, others are inside the building.
- Some cameras have predefined scanning movement possibilities, other cameras look in a fixed direction all the time.
- The width of a camera view is 50 degrees.
- The camera costs are $500 per camera per year. These costs are caused by maintenance and camera value deduction. Costs of surveillance monitors, wires and data storage devices are neglected (or considered loaded into the camera cost).

2.5 Optimization Heuristics

Because of the complexity of the objective function evaluation and the large search space size, heuristics are considered for optimization. Heuristics require a solution encoding, a method to evaluate solutions, a method to move to or to identify new candidate solutions and a termination mechanism.

2.5.1 Security Configuration Representation and Evaluation

The problem is described as a binary problem, where each solution consists of two strings with binary digits, one string lists possible camera locations while the second string lists possible guard paths. For each predefined candidate camera location or guard path, there is one digit. If the digit is set to 0 this means that the candidate camera location or guard path is not used, if the digit is set to 1 the candidate camera location or guard path is used in the security configuration. Summing over the camera string yields the number of cameras while the same is true for guards when summing over the guard string.

Each solution is evaluated by the mean of three replications of the simulation. If the mean of these three indicates infeasibility (does not meet the detection/apprehension constraint), the solution is discarded. Otherwise, the solution is considered for inclusion in the population. A more precise simulation set of 30 replications is done for each solution which has a higher fitness than the worst solution in the current population and has a fitness which is not more than 10% lower than the fitness of the best solution in the population. The mean of these 30 replications is used as the evaluation. Thirty replications are chosen as that is an accepted lower bound on "large sample size", where normality of the mean can be assumed. The reason for this two step evaluation is that the simulation takes quite a bit of CPU time, and this method conserves it.*

* For a number of solutions, the 30 replications were also compared with 600 replications. It was ascertained that 30 provided the same results as 600, and therefore for this simulation, was deemed sufficient for optimization.

2.5.2 The Need for an Effective Heuristic

There are 41 candidate camera locations and 17 candidate guard paths resulting in a 58 dimensional binary solution space. The number of cameras and guards used in the security configuration is also determined by the heuristic, therefore there are $2^{58} \approx 2.9 \times 10^{17}$ solutions. Because there is stochastic behavior in the simulation (for example, door or window opening time), each solution has to be tested many times to obtain the distribution of results. Because of this and because the highly detailed simulation is computationally expensive, there is a need for a very effective heuristic.

Two popular heuristics inspired by nature are combined into a new hybrid algorithm, the hybrid genetic ant optimization (HGAO). HGAO combines properties of genetic algorithm (GA) and ant colony optimization (ACO), which for this problem is more effective than either heuristic alone. In general ACO converges faster than GA, because ACO especially depends on local information (pheromone level of each single digit) and GA especially relies on global information (fitness of entire solution). Specifications of the GA and the ACO are given in Sections 2.5.2 and 2.5.3, respectively.

2.5.3 GA

In the genetic algorithm strings can been seen as chromosomes and digits can be seen as genes. For general information on GA, see Ananda et al. [1] for pseudo code on the application of the GA in this chapter, see Figure 2.2. In this chapter, initializing the GA is done as follows: First, random initial solutions are created. This is done by choosing a random number between 0 and 1 for each initial solution, then for each bit in the string, another random number is chosen between 0 and 1. At each bit place, if the random number is larger than string random number, a 1 is assigned for that bit, otherwise a 0 is assigned. After initialization, a genetic algorithm consists of four stages; selection, recombination, mutation and updating.

Selection: The two parents that are used for creating the next child(ren) can be selected in various ways. In this chapter single round tournament selection is used. For each parent two individuals from the population are chosen at random. The fittest of these two individuals will be one of the next parents. In this chapter two parents only create one child.

Recombination: In this chapter equal probability gene cross-over (also called uniform crossover) is used. For each gene, there is a 50% chance that the gene (0 or 1) of the first parent is used and a 50% chance that the gene of the second parent is used.

Mutation: In this chapter gene flipping probability is used. This means that for every gene the probability that the gene flips (either from 0 to 1 or from 1 to 0) is defined. The probability is set to 0.05. Using this probability for gene flipping results in some strings with no mutations and some strings with one or several mutations.

Updating: Usually whenever an entire population has been created and tested, the fittest solutions are selected following a certain strategy and these solutions create a new generation, termed a general

```
Pseudo code GA
Repeat
      Create one random individual
      Evaluate solution
      If feasible
            Add to population
Until (number of individuals = max population size)

Repeat
      Select 2 parents by tournament selection
      Apply genetic operators to generate new individual
      Evaluate solution
      If feasible & fit enough
            Add to population and remove worst individual
Until too many steps without improvement OR max calculation time reached
```

FIGURE 2.2 Pseudo code of the genetic algorithm.

GA. However, because of the computational effort of evaluation, an incremental GA is used here. After being created (by two parents), the child will only be added to the population if its fitness is higher than the fitness of the solution with the lowest fitness in the population. Solutions that are not feasible with respect to the constraints described in Section 2.3 will be discarded immediately.

2.5.4 ACO

In the ant colony optimization algorithm, strings can be seen as paths and each digit can be seen as a dimension. So if a string consists of n digits, the problem is n-dimensional. For each dimension there is a pheromone concentration. This concentration is a real number, the probability that an ant walks into this dimension (that, the probability that the search moves in to this solution), depends on this pheromone concentration. The pheromone levels are initialized by initially setting all values to 0.5 (or, to a random walk). For general information on ACO, see Kong and Tian [7], for pseudo code on the application of the GA in this chapter, see Figure 2.3.

An ant colony algorithm consists of three stages; path picking, pheromone increase and pheromone decrease.

Path picking: In this chapter double probabilistic path picking is used. First the number of digits that will be set to 1 in the next solution will be determined via a probability. The mean of this probability equals the number of digits that were set to 1 in the corresponding string in the previous solution. Once the numbers of digits that will be set to 1 in each string (guard string and camera string) are determined, the actual digits that are set to 1 are chosen in a probabilistic way using the pheromone concentration.

Pheromone increase: For cameras, whenever a camera detects an intruder, the pheromone level for its corresponding digit is raised. For a guard path, the pheromone level is increased when s/he actually catches an intruder. In addition, the pheromone level of a guard increases if the guard detects an intruder on the surveillance monitor. Note that even for solutions that are infeasible with respect to the constraints given in Section 2.3 the pheromone levels can increase.

Pheromone decrease: The ACO in this chapter uses min/max criteria. Whenever a pheromone level exceeds a maximum value, the pheromone levels for all genes on that string will be scaled down by the same fraction in such a way that the highest pheromone level equals the maximum value. Whenever a pheromone level comes below a minimum value, this pheromone level is set to the minimum value. This prevents pathological behavior of the ACO search.

2.5.5 HGAO

In HGAO, each new solution is created either by GA or by ACO (Figure 2.4). The idea behind this is that whenever an ACO solution is created, it will be close to the previous solution in the solution space. If this results in lower fitness, the next solution will probably be created using GA. However, if the ACO

```
Pseudo code ACO
For every bit
      Set initial pheromone concentration
Repeat
      Create one random individual
      Evaluate solution
      Update pheromone levels
Until n random individuals have been created

Repeat
      Create new individual based on pheromone levels
      Evaluate solution
      Update pheromone levels
Until too many steps without improvement OR max calculation time reached
```

FIGURE 2.3 Pseudo code of the ant colony optimization.

```
Pseudo code HGAO
For every bit
      Set initial pheromone levels
Set initial Xp (Xp = probability that next individual is created by ACO
strategy)
Repeat
      Create one random individual
      Evaluate solution
      Update pheromone levels
      If feasible
            Add to population
Until (number of individuals = max population size)
Repeat
      If (random number between 0 and 1 < Xp)
            Create new solution based on pheromone levels
            Evaluate solution
            Update Xp
      Else
            Apply genetic operators to generate new solution
            Evaluate solution

      Update pheromone levels
      If feasible & fit enough
            Add to population and remove worst individual
Until too many steps without improvement OR max calculation time reached
```

FIGURE 2.4 Pseudo code of the hybrid genetic ant optimization.

solution results in a higher fitness, the probability that the next solution will be created by ACO again is high. This provides a balance between the strengths and successes of each heuristic.

Which one of the two heuristics (GA or ACO) is used for creating the next solution depends on the value of variable X_p in the HGAO heuristic. X_p is a real value ranging from X_{min} ($X_{min} \geq 0$) to X_{max} ($X_{max} \leq 1$). The value of X_p directly represents the probability that the next solution will be created using ACO. If a solution is not created by ACO, it is created by GA. X_{min} and X_{max} are static real values, they are both set before the optimization starts. X_p is a real value that changes during the optimization. For this work X_{min} was set to 0.2 and X_{max} was set to 1. This means that at least 20% of the solutions are created using the ACO approach and in the other extreme case all solutions are created using ACO approach.

There are many ways to update X_p. In this work a simple decision strategy is chosen—X_p is only updated after testing an ACO solution. If the ACO solution gave a good result (feasibility and high fitness), the value of X_p will be increased by X_{add}, so the probability that the next solution will be created using ACO will be higher. If the ACO solution gave a bad result (infeasibility or low fitness), the value of X_p will be decreased by $X_{substract}$, so the probability that the next solution will be created using ACO will be lower. X_{add} and $X_{substract}$ are set to the relatively high values 0.4 and 0.2, respectively, so X_p can fluctuate rapidly. The idea behind this decision strategy is that the GA effectively scans the global search space while the ACO part specializes in local optimization.

2.6 Results

The optimization methods and the simulation have been written in such a way that various security optimization problems can be solved. In this section results of two different optimization problems are given. Results of the GA and of the HGAO will be compared. Furthermore an in-depth look at the HGAO heuristic's behavior will be given.

The optimization problem described in Section 2.6.1 is large and complicated. This problem is where all possibilities of the simulation and optimization are tested. The optimization problem in Section 2.6.2 is a smaller one and this problem is chosen to get an overview of the effects of setting constraints and of using alternative numbers of guards. A population of 30 is used by each algorithm for each problem.

2.6.1 Test 1: Unconstrained Full Optimization

The goal of this optimization is to find the lowest total cost security configuration. Both cameras and guards can be used. No constraints are used, that is, no minimum levels of detection or apprehension are set. The reason for optimizing this problem is that it is a large and complicated problem. It would be difficult to find a good security configuration for this problem using only common sense and intuition. Forty intruder scenarios are considered, 41 possible camera locations and 17 possible guard paths. The optimization is terminated when number of intruder runs reaches three million, resulting in testing approximately 14K different security designs and taking about 48 hours of desktop PC computing time. Note that the theoretic search space is $2^{(40+41+17)} = 7.9 \times 10^{28}$.

2.6.1.1 Results

The same problem was optimized using GA and HGAO. Both the GA and the HGAO are run for approximately 14,000 iterations, which takes approximately two days (each algorithm) on a current desktop computer. Recall that the simulation itself is the greatest consumer of computational resources. Figure 2.5 shows a typical run of each algorithm as computational time (and thus, iterations) increase (note that the *x* axis is log scale).

One can see that both algorithms have similar performance. In the beginning they perform exactly the same with respect to the total costs of the best solution so far. This is due to the fact that initial solutions for the GA are created with the same seeding in both methods. The GA has a population size of 30, so it is seeded with 30 individuals.

In the end, HGAO finds a slightly better solution than the GA (see Table 2.1). Interestingly, each optimization ends up in a different optimum. Both find that the only guard that should be used is the guard that watches the surveillance monitors all the time (except when an intruder is detected, in which case the guard leaves the monitors to attempt to apprehend the intruder). Both optimizations find that 27 of the 41 candidate cameras should be used. Cameras specified for the upper floors were chosen to view the elevator or stairwells, as these are the only access points from below. Each method finds a different camera configuration (see Figure 2.6 for specifics). The security configuration of the HGAO results in a 0.2% higher detection probability and a 0.5% higher apprehension probability than the security configuration found by the GA. Due to these probabilities the costs induced by intruders are 3.6% lower for the HGAO configuration. A little more than half of the total costs are security costs (guards and cameras)

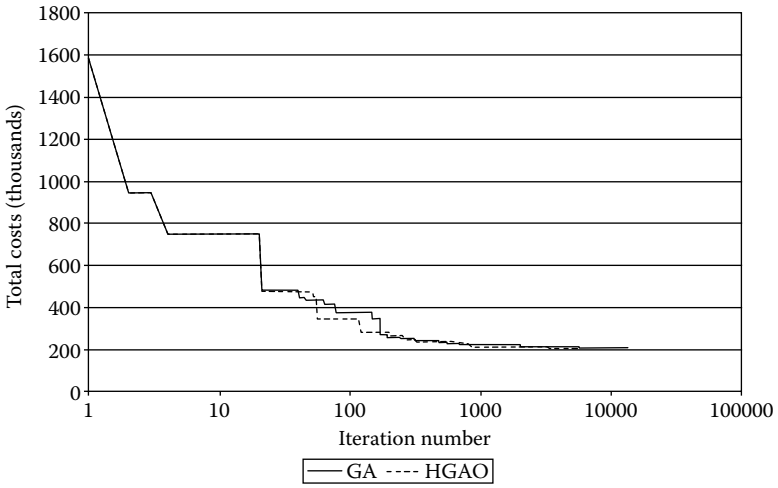

FIGURE 2.5 Total costs of the best solution of each optimization approach as search progresses.

TABLE 2.1 Results of Optimizations with GA and with HGAO

	GA	HGAO
Detection probability	0.967	0.969
Probability of getting caught	0.956	0.961
Number of guards used	1	1
Number of cameras used	27	27
Guard costs	$96,720-(46.1% of total costs)	$96,720-(47.6% of total costs)
Camera costs	$13,500-(6.4% of total costs)	$13,500-(6.7% of total costs)
Intruder caused costs	$99,720-(47.5% of total costs)	$92,572-(45.6% of total costs)
Total costs	$209,904	$202,792

while the rest are costs incurred by break ins. In the lowest cost security configuration about 96% of the intruders get caught.

To investigate whether HGAO finds the slightly better solution because of stochastic behavior in the simulations or because the HGAO is a better heuristic for this problem, a more in depth analysis is done in Section 2.6.1.2.

2.6.1.2 In-depth Look at the HGAO Behavior

The idea behind the decision strategy of HGAO is that the ACO part will efficiently search local areas of promising solutions (see Sections 2.5.2 and 2.5.5). This means that in most cases the ACO part should find (near-) optimal solutions for each optimum. If this is the case, the majority of solutions that are good enough to be added to the population should be created by ACO. Furthermore, the best solution that is found at any iteration is most likely to be found by ACO.

However, it turns out that for this problem only 13.3% of the solutions that are fit enough to be added to the simulation are ACO solutions. All other solutions are created by the GA. Of the best solutions so far, only 16.0% were found by the ACO part while the total fraction of solutions created by the ACO was 34.1%. From these statistics, one can infer that an ACO solution is less likely to be a good solution than a GA solution. This outcome contradicts the expected behavior of the HGAO and the fact that HGAO slightly outperformed GA can be attributed to the stochastic behavior of both methods.

2.6.2 Test 2: Constrained Optimization with Guards Only

The goal of the second test is to consider the optimization problem with no cameras. This might be useful in conflict situations where military units move often or in emergency situations (such as just after a natural or manmade disaster) where no camera infrastructure is available. In this case, a constraint on the minimal fraction of intruders that gets caught is set to various values. The trade offs in number of guards, security and total costs and fraction of apprehension will be examined. Up to 17 candidate guards (with their predefined paths) are considered, resulting in a search space of $2^{17}=131$K. The optimization termination criterion was 300K intruder runs, equivalent to 1597 possible security designs (for each chosen level of security).

Both GA and HGAO found the same security configurations for each optimization, regardless of constraint value. Calculation times were also similar. Therefore no separate results of GA and HGAO are given. Note that in some cases the optimal security configuration is not located on a constraint, but is located in the feasible area slightly away from the constraint. If for instance h_1 is set to 0.95, at least 95% of the intruders must get caught by a guard. Due to the integer number of guards, the outcome might be that if h_1 is set to 0.95 the probability that an intruder gets caught is 0.97. In Figures 2.7 and 2.8 the probabilities of getting caught are not the constraint value, they are the actual probability belonging to the security configuration found. Both figures do not provide results for lower probabilities than 0.899, because if no constraints (or very loose constraints) are used, the best solution (i.e. lowest total costs) has

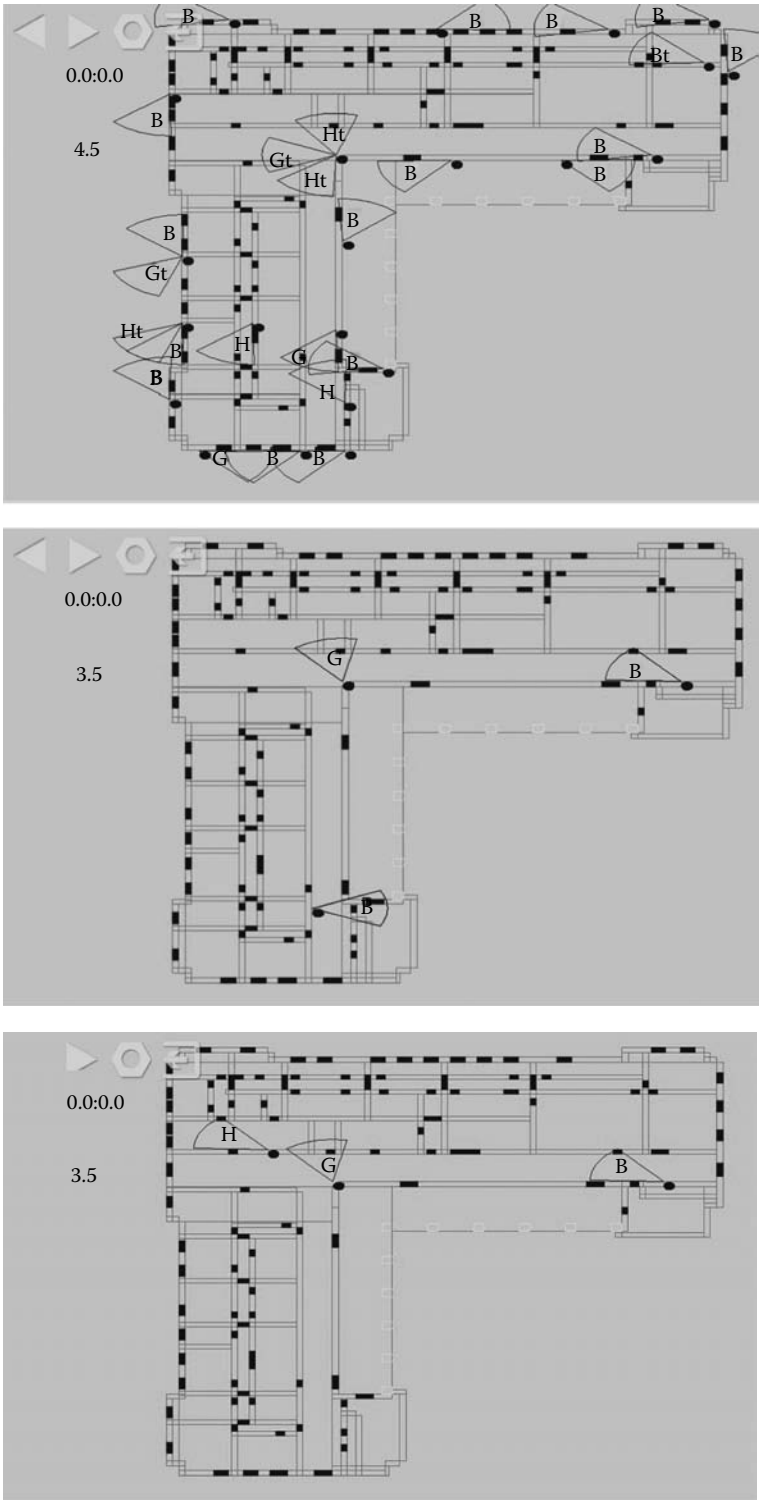

FIGURE 2.6 Optima from HGAO and GA (ground, second and third floors from top to bottom). H: camera only used in HGAO; G: camera only used in GA; B: camera used in both; t: camera can turn, it scans a larger surface.

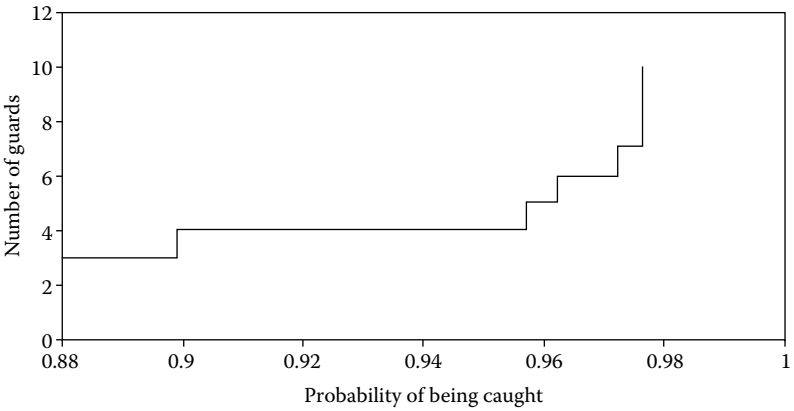

FIGURE 2.7 Number of guards needed versus probability of being caught.

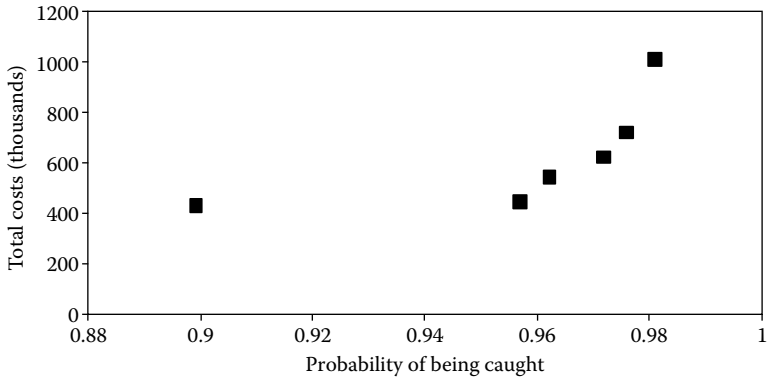

FIGURE 2.8 Total cost versus probability of being caught.

a probability of being caught of 0.899. Furthermore, regardless of constraint level, the final solution of the optimization never has less than three guards.

When more guards are used, more intruders get caught, so the costs caused by intruders decrease. However, when more guards are used, the guard costs increase. From Figures 2.7 and 2.8 one can conclude that for strict constraints, the more guards used, the higher the total costs are. This means that the major part of the total costs must consist of guard salaries. That this is indeed the case can be seen in Table 2.2. This table shows the total costs and the fraction of costs caused by guard salaries.

For the optimal case with respect to total costs, three of the 17 candidate guards are required if no constraints are set. The number of guards needed to obtain a certain probability that an intruder gets caught seems to increase exponentially until it reaches a level of 10 guards (with 98.1% probability of apprehension). After ten guards, no further improvement in the probability of being caught can be made. This is not caused by a flaw in the simulation nor in the optimization heuristic, rather it is due to the fact that the predefined candidate guard patrol paths are not above 98.1% effective. When all 17 possible guards are used, still 98.1% of the intruders get caught. To increase that percentage, more and/or different candidate guard paths would need to be defined. Figures 2.9 and 2.10 show two of the optimal configurations—one with three guards and one with six guards. The three guard solution consists solely of paths around the exterior of the building. With six guards, two more are added to the ground floor, both inside the building, while one is stationed on the second floor.

TABLE 2.2 Costs for Various Probabilities of being Caught for an Intruder Found by HGAO

Probability of being Caught	Guard Salary Costs	Intruder Caused Costs*	Total Costs
0.899 (optimal with respect to total costs)	$290,160 (67.5% of total costs)	$139,566 (32.5% of total costs)	$429,726
0.957	$386,880 (86.8% of total costs)	$58,838 (13.2% of total costs)	$445,718
0.962	$483,600 (89.2% of total costs)	$58,525 (10.8% of total costs)	$542,125
0.972	$580,320 (92.1% of total costs)	$49,806 (7.9% of total costs)	$630,126
0.976	$677,040 (93.0% of total costs)	$51,012 (7.0% of total costs)	$728,052
0.981	$967,200 (95.6% of total costs)	$44,825 (4.4% of total costs)	$1,012,025

* Note that the costs caused by the intruder for a probability of being caught of 0.976 are slightly higher than for a probability of 0.972. This is caused by stochastic behavior of the simulation.

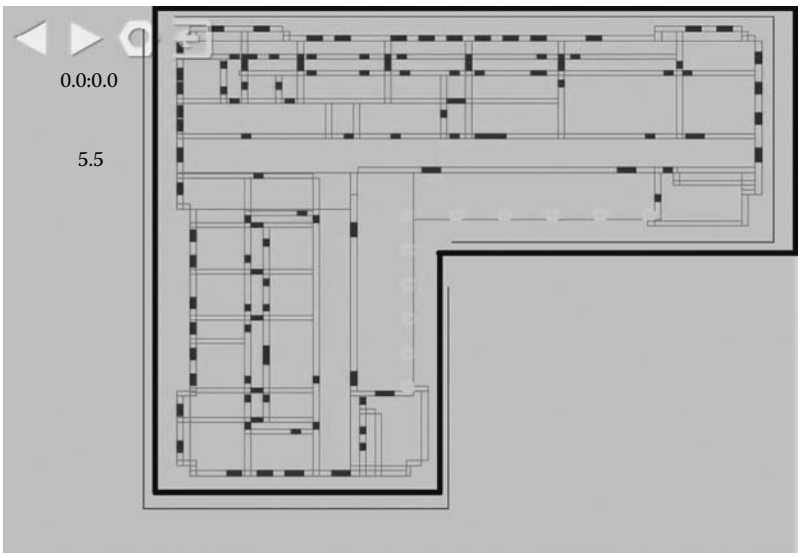

FIGURE 2.9 Best solution with three guards (probability of apprehension = 89.9%). Thick line: guard walks in circles. Other lines: guards walk over this line back and forth.

2.7 Discussion

This chapter described the use of heuristic optimization coupled with detailed stochastic simulation to design physical security systems. Because of the computational effort needed for the simulation, the optimization routine needs to be efficient as it searches a very large combinatorial space. The heuristics described have the advantage that they can be terminated at any point and still return good solutions, and in fact, almost always experience diminishing returns over search time with respect to the objective function (see Figure 2.5 for example). Furthermore, the objectives and constraints are quite flexible in this method so that various problems could be posed and solved using the same simulation and the same optimization framework. Trade offs between alternative security levels and the investment needed can be ascertained.

There are drawbacks to the approach. One is that candidate guard patrol paths, camera locations and details, and possible intruder scenarios have to be defined in advance. Defining them can be time consuming as well as conceptually challenging. The optimization only uses these predefined data, therefore the security design consists of elements among the prespecified ones. A second drawback is the optimization itself. Heuristics are not guaranteed to find optima, and it is even difficult to know how to close to optimal the results are. The heuristics herein discard infeasible solutions, that is those which do not

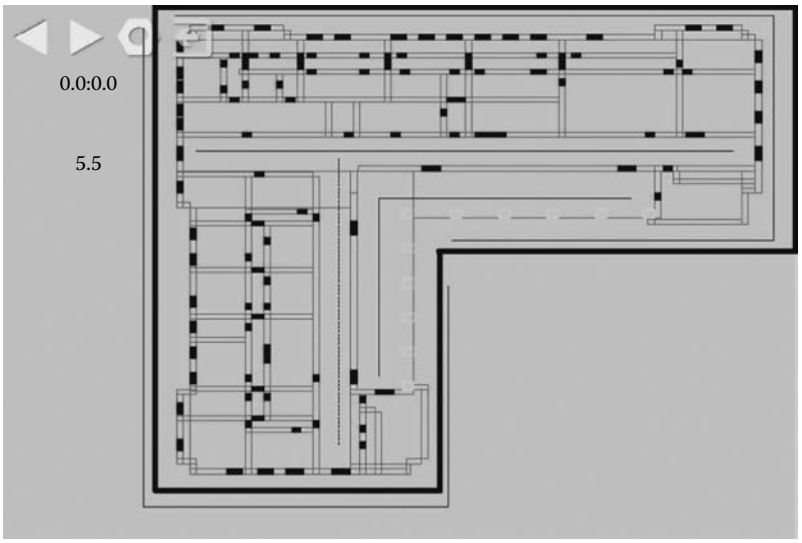

FIGURE 2.10 Best solution with six guards (probability of apprehension = 97.2%). Thick line: guard walks in circles. Other lines: guards walk over this line back and forth. Dotted line: guard walks on second floor. On all other lines the guard walks on the ground floor (or outside).

meet the security constraint. It may be advantageous to include these in the population as a source of valuable information for the search mechanism. A final drawback is the simulation model. Adding more detail to it can make things even more realistic. Currently, intruder behavior is independent of security (that is, guards and cameras). This is clearly not the actual case and the simulation would be improved by adding the facility for guards, cameras and intruders to interact. This might be done by creating a network of arcs and nodes and let intruders and guards decide which arc to take at each node on the fly, based on one or several decision variables. Guards should be able to notice certain state changes (for example opening or closing doors) that have occurred, which would indicate an intrusion without seeing the intruder. Guards could then act upon this observation.

This work could be straightforwardly extended by including a wider range of intruder scenarios and studying the scenarios with alternative security set ups in a rigorous manner. A design of experiments might be devised to systematically analyze outcomes and identify significant main effects and interactions. Another straightforward extension is to alter some of the assumptions regarding intruder and guard behavior. Examples are to relax the one intruder at a time assumption or to model behavior in which a guard chases an intruder. These would add realism to the system but would take additional simulation coding and verification, as well as additional computational resources during scenario testing.

However, as a first step in combining detailed simulations with optimization for design of secure facilities, this work offers value and promise. The method is not designed to be "black box"; rather it is designed to supplement human decision making to effect rigorous and defensible decisions. Certainly, when designing for securing, it is vital to know to the best degree possible the ramifications of alternative designs, and to identify all superior designs. This method accomplishes these goals effectively, and should thereby be a valuable tool for security decisions.

References

1. Ananda Rao, M., Srinivas, J., Murthy, B.S.N. 2004. Damage detection in vibrating bodies using genetic algorithms. *Computers and Structures*, 82, 963–968.

2. Bu, T-M., Yu, S-N., Guan, H-W. 2004. Binary-coding-based ant colony optimization and its convergence. *Journal of Computer Science and Technology English Language Edition,* 19(4), 472–479.
3. Cavalier, T.M., Conner, W.A., Del Castillo, E., Brown, S.I. 2007. A heuristic algorithm for minimax sensor location in the plane. *European Journal of Operational Research,* 183, 42–55.
4. Hamel, D., Chwastek, M., Farouk, B., Dandekar, K., Kam, M. 2006. A computational fluid dynamics approach for optimization of a sensor network. *IEEE International Workshop on Measurement Systems for Homeland Security, Contraband Detection and Personal Safety IEEE,* Cat No. 06EX1330C, 38–42.
5. Jones, D.A., Turnquist, M.A., Nozick, L.K. 2005. Simulation of imperfect information in vulnerability modeling for infrastructure facilities. *Proceedings of the 2005 Winter Simulation Conference,* Orlando, Florida, eds Kuhl, M.E., Steiger, N.M., Armstrong, F.B., Joines, J.A., The Society for Computer Simulation International (SCS), San Diego, CA, pp. 965–971.
6. Jordan, S.E., Snell, M.K., Madsen, M.M., Smith, J.S., Peters, B.A. 1998. Discrete-event simulation for the design and evaluation of physical protection systems. *Winter Simulation Conference Proceedings,* Washington, DC, The Society for Computer Simulation International (SCS), San Diego, CA, pp. 899–907.
7. Kong, M., Tian, P. 2006. A new ant colony optimization applied for the multidimensional knapsack problem. *Lecture Notes in Computer Science,* 4247, pp. 142–149.
8. Nahas, N., Nourelfath, M. 2005. Ant system for reliability optimization of a series system with multiple-choice and budget constraints. *Reliability Engineering and System Safety,* 87, 1–12.
9. Olario, S., Nickerson, J.V. 2005. Protecting with sensor networks: perimeters and axes. *Military Communications Conference (MILCOM) 2005, Atlantic City, New Jersey,* 3, IEEE, New York, pp. 1780–1786.
10. Smith, J.S., Peters, B.A., Jordan, S.E., Snell, M.K. 1999. Distributed real-time simulation for intruder detection system analysis. *Proceedings of the 1999 Winter Simulation Conference,* Phoenix, Arizona eds Farrington, P.A., Nembhard, H.B., Sturrock, D.T., Evans, G.W., The Society for Computer Simulation International (SCS), San Diego, CA, pp. 1168–1173.
11. Ustun, V., Yapicioglu, H., Gupta, S., Ramesh, A., Smith, J.S. 2005. A conceptual architecture for static features in physical security simulation. *Proceedings of the 2005 Winter Simulation Conference,* Orlando, Florida, eds. Kuhl, M.E., Steiger, N.M., Armstrong, F.B., Joines, J.A., The Society for Computer Simulation International (SCS), San Diego, CA, pp. 958–964.
12. Yeo, B.-K., Lu, Y. 1999. Array failure correction with a genetic algorithm. *IEEE Transactions on Antennas and Propagation*, 47(5), 823–828.