Real-time Allocation of Defensive Resources To Rockets, Artillery, and Mortars

Fredrik Johansson
Informatics Research Centre
University of Skövde, Sweden.
fredrik.johansson@his.se

Göran Falkman
Informatics Research Centre
University of Skövde, Sweden.
goran.falkman@his.se

Abstract – *The protection of defended assets such as military bases and population centers against ballistic weapons (e.g. rockets and mortars) is a highly relevant problem in the military conflicts of today and tomorrow. In order to neutralize threats of this kind, they have to be detected and engaged before causing any damage to the defended assets. We propose algorithms for solving the resource allocation problem in real-time, and empirically investigate their performance using the open source testbed SWARD. The results show that a particle swarm optimization algorithm produce high quality solution for small-scale problems, and that a genetic algorithm yields the best solutions for the largest tested problem instances.*

Keywords: Air defense, resource management, weapon allocation.

1 Introduction

A severe threat encountered in many international peacekeeping and peace forcing operations is that of rockets, artillery, and mortars (RAM) fired by insurgents towards military bases, troops, and other assets. Attacks like these have cost many human lives in places like Iraq and Afghanistan during recent years. Similar attacks are faced by civilians in some parts of Israel on a regular basis, where so called Katyusha and Qassam rockets are fired against Israeli population centers such as Sderot and Ashkelon. Asymmetrical threats like these have caused an increased interest in systems for detecting and tracking incoming RAM before they hit their intended targets. The detection and tracking of RAM makes it possible to estimate the point of impact, so that any troops or civilians in the impact area can be alerted. However, such a warning is not always enough, due to very quick course of events, and that buildings and infrastructure will be destroyed no matter what how early warnings are, given that active countermeasures are not taken. Hence, one would like to destroy incoming RAM before they hit their intended targets (and before they risk causing collateral damage upon destruction). Systems for detecting, tracking, and engaging RAM are often referred to as Counter Rocket, Artillery, and Mortar (C-RAM) systems. Examples of such systems are Centurion (a ground based version of the Phalanx CIWS), Skyguard (a laser based system), and Iron Dome (a system using intercepting missiles).

When faced with many simultaneous threats, it is unlikely that the defenders can take action against all incoming threats, since there often are fewer firing units available than there are threats. Even when this is not the case, a problem is to know which firing unit to use against which threat in order to optimize the survivability of the defended assets. This can be described as a resource allocation problem, known as the weapon allocation problem [5] within the field of operations research. Unfortunately, the allocation of defensive weapon resources to targets has been shown to be NP-complete [11]. To complicate matters, the allocation problem studied in this paper has to be solved in a short amount of time, due to the high speed of incoming threats, and the short-range they often are fired from. Therefore, empirical results for how weapon allocation algorithms perform on problem instances of various size are needed. Such results are lacking for the weapon allocation problem studied in this paper (the static asset-based weapon allocation problem), which is the reason for why we empirically test three different heuristic weapon allocation algorithms and compare the quality of their generated solutions. The tested algorithms are a genetic algorithm, a particle swarm optimization algorithm, and a greedy maximum marginal return algorithm.

The rest of this paper is structured as follows. In Section 2, we present the static asset-based weapon allocation problem, which is a suitable optimization model when the impact area of a threat can be assumed to be known. We also briefly present its target-based counterpart. In Section 3, we describe related work and present three different heuristic algorithms for solving the static asset-based weapon allocation problem. These algorithms have been empirically tested on some small-scale and larger-scale scenarios, and the experiments and results are presented in Section 4. Finally, we conclude the paper and present thoughts for future work in Section 5.

2 Weapon allocation

Informally, weapon allocation (often also referred to as weapon assignment or weapon-target allocation) can be defined as the reactive assignment of defensive weapon resources (firing units) to engage or counter identified threats (e.g., aircrafts, air-to-surface missiles, and rockets) [15]. More formally, the weapon allocation problem can be stated as a non-linear optimization problem in which we aim to allocate firing units so as to minimize the expected total value of the targets, or, alternatively, to maximize the expected survivability of the defended assets. These alternative views are referred to as target-based (weighted subtractive) defense and asset-based (preferential) defense, respectively. The asset-based formulation demands knowledge of which targets that are headed for which defended assets and thereby assumes a high level of situation awareness [12]. Therefore, the static asset-based weapon allocation problem formulation is suitable for problems involving defense against ballistic weapons, while the target-based formulation is more appropriate when the intended aim of the targets are not known [14]. In Section 2.1 we describe the static asset-based weapon allocation problem in detail, and provide a brief presentation of the static target-based weapon allocation problem in Section 2.2.

2.1 The static asset-based weapon allocation problem

When presenting the static asset-based weapon allocation problem, the following notation will be used:

- $|\mathbf{A}| \triangleq$ number of defended assets.
- $|\mathbf{W}| \triangleq$ number of firing units.
- $|\mathbf{T}| \triangleq$ number of targets.
- $\omega_j \triangleq$ protection value of defended asset A_j.
- $P_{ik} \triangleq$ probability that firing unit W_k destroys target T_i if assigned to it.
- $\pi_i \triangleq$ probability that target T_i destroys the asset it is aimed for.
- $\mathbf{G}_j \triangleq$ the set of targets aimed for defended asset A_j.
- $x_{ik} = \begin{cases} 1 & \text{if firing unit } W_k \text{ is assigned to target } T_i, \\ 0 & \text{otherwise.} \end{cases}$

In the static asset-based weapon allocation problem, each offensive target is assumed to be aimed at a defended asset, where each defended asset is associated with a protection value ω_j. Each target has an associated lethality probability π_i, indicating the probability that T_i destroys the defended asset it is aimed for, given that it is not successfully engaged. This probability depends on the accuracy of the targets as well as the nature of the defended assets [6]. As can be seen, we are assuming that such probabilities only are target dependent, i.e., we do not take the type of the defended asset into consideration. The defenders are equipped with firing units, where each pair of firing units and targets is assigned a kill probability P_{ik}. Now, the objective of the defense is to allocate the available firing units so as to maximize the expected total protection value of surviving targets [5]:

$$\max J = \sum_{j=1}^{|\mathbf{A}|} \omega_j \prod_{i \in \mathbf{G}_j} (1 - \pi_i \prod_{k=1}^{|\mathbf{W}|} (1 - P_{ik})^{x_{ik}}), \qquad (1)$$

subject to:

$$\begin{gathered} \sum_{i=1}^{|\mathbf{T}|} x_{ik} = 1, \ \ \forall k, \\ x_{ik} \in \{0, 1\}, \ \ \forall i \forall k. \end{gathered} \qquad (2)$$

A solution to a static weapon allocation problem can be represented as a matrix of decision variables

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1|\mathbf{W}|} \\ x_{21} & x_{22} & \dots & x_{2|\mathbf{W}|} \\ \vdots & \vdots & x_{ik} & \vdots \\ x_{|\mathbf{T}|1} & x_{|\mathbf{T}|2} & \dots & x_{|\mathbf{T}||\mathbf{W}|} \end{bmatrix}. \qquad (3)$$

Such a solution is feasible if it fulfills the constraints given in Equation 2, i.e., that the entries of each column in Equation 3 sum to one. For a problem instance consisting of $|\mathbf{T}|$ targets and $|\mathbf{W}|$ firing units, there are $|\mathbf{T}|^{|\mathbf{W}|}$ feasible solutions.

From the solution of the static asset-based weapon allocation problem, we can discover which of the defended assets that should be protected, and in which way each of the defended assets should be protected (preferential defense). A few assumptions are made in the static asset-based weapon allocation formulation. Firstly, all firing units have to be assigned to targets, as indicated in the constraint given in Equation 2. Moreover, all the firing units have to be assigned simultaneously, i.e., we can not observe the outcome of some of the engagements before a remaining subset of firing units are allocated. This is what is meant by *static* weapon allocation, as opposed to *dynamic* weapon allocation. The static formulation makes sense for the problem domain studied here, since the high speed of short-range RAM does not allow for several engagement cycles. We also assume that an engagement will not affect other engagements (e.g., that a firing unit can destroy another target than it is allocated to, or that targets can destroy other assets than they are aimed for). Without the last assumption, the geometry of the problem must be taken into account, creating an extremely complex problem. We also ignore the risk of collateral damage to the protected area when intercepting the targets.

Despite the assumptions, there is a combination of factors that make the static asset-based weapon allocation problem hard to solve. Firstly, the objective function given in Equation 1 is *non-linear*, so that well-known linear programming techniques such as the simplex algorithm can not be used to

solve the problem. Secondly, the problem is *discrete*, since it only allows for integer valued feasible solutions due to the second constraint in Equation 2 (i.e., fractional allocations are not possible). In general, this kind of integer programming problems are hard to solve. Thirdly, the problem is *stochastic*, due to kill probabilities and lethality probabilities not equal to zero or one. This non-determinism further complicates the problem. Fourthly, it is not unusual with *large-scale* problem instances, i.e., problems consisting of a large number of firing units, defended assets, and/or targets. The asset-based formulation can be shown to be a generalization of the static target-based weapon allocation formulation [5] presented in Section 2.2. The NP-completeness of the static target-based weapon allocation problem was established in [11], and hence, we can conclude that the static asset-based weapon allocation problem is *NP-complete* as well [5]. These properties taken together show that finding good solutions in real-time to the static asset-based weapon allocation problem is indeed a very hard problem, and according to [6], rule out any hope of obtaining efficient optimal algorithms.

2.2 The static target-based weapon allocation problem

Using the same notation as in Section 2.1, but with the additional definition: $T_i \triangleq$ target value of target T_i, we can define the static target-based weapon allocation problem as:

$$\min F = \sum_{i=1}^{|\mathbf{T}|} V_i \prod_{k=1}^{|\mathbf{W}|} (1 - P_{ik})^{x_{ik}}, \tag{4}$$

subject to the constraints given in Equation 2. This formulation is included in this paper since the greedy algorithm presented in Section 3.4 relies upon it. For a more thorough presentation of the static target-based weapon allocation problem, see e.g. [8] and [5].

3 Algorithms for weapon allocation

In this section, we describe a number of algorithms for static weapon allocation. Section 3.1 gives an overview of algorithms that earlier have been suggested for the target-based formulation of the problem, as well as the less researched asset-based formulation. A genetic algorithm is presented in Section 3.2, a particle swarm optimization algorithm in Section 3.3, and a greedy algorithm is presented in Section 3.4. We have earlier applied these algorithms to air defense situations in which a target-based objective function must be used [8], but their performance on situations involving RAM (i.e., where an asset-based objective function can be used) is previously unknown.

The algorithms suggested in this paper share the same kind of representation, in which a solution is represented as a vector of length $|\mathbf{W}|$. Each element k in the vector points out the target T_i to which the weapon is allocated. As an example of this, the vector $[2, 3, 2, 1]$ represents a solution in which W_1 and W_3 are allocated to T_2, W_2 is allocated to T_3, and W_4 is allocated to T_1.

3.1 Related work on weapon allocation

An intuitive and natural way to try to solve static weapon allocation problems is to go through all possible allocations one by one, and return the solution that maximizes (asset-based defense) or minimizes (target-based defense) the objective function. A drawback with such an exhaustive search algorithm is the bad scaling. Despite the computational power available today, only small-scale problems can be solved in real-time, using such an algorithm. However, there is no reason for not using an exhaustive search algorithm when the problem instance is small enough, since it always returns the optimal solution. How large static asset-based problem instances that can be solved in real-time by exhaustive search is empirically investigated in Section 4.1.

Due to the bad scaling of exact algorithms, there is often no other option than to rely on heuristic algorithms, i.e., approximate algorithms that cannot guarantee to find the optimal solution, but are likely to return good solutions in a significantly reduced amount of time [1]. Well-known examples of heuristic approaches for different combinatorial optimization problems are greedy algorithms, genetic algorithms, simulated annealing, taboo search, ant colony optimization, and particle swarm optimization. All of the algorithmic approaches mentioned above have been applied to the static target-based formulation of the weapon allocation problem (see e.g., [10, 16, 9]), but to the best of our knowledge, it is not known how such heuristic algorithms perform on the asset-based formulation of the problem. In general, most of the available literature on weapon allocation considers the target-based problem, rather than the asset-based problem [6].

A few heuristic algorithms for asset-based weapon allocation are suggested by Metler and Preston in [13]. One approach suggested in [13] is to approximate the asset-based problem with its target-based counterpart and use a greedy maximum marginal return algorithm to return a solution that can be used as an approximative solution to the asset-based problem. Another suggested approach is to use the solution returned from the greedy maximum marginal return algorithm and to apply local search on the solution so that the target allocated by one weapon can be swapped to the target allocated by another weapon, and vice versa. In [2], a genetic algorithm combined with local search is suggested for a dynamic version of the asset-based weapon allocation problem. It is shown that local search improves the results compared to use the genetic algorithm without local search, but that the computational time needed is increased. The effect of real-time requirements on the algorithms are not tested. A simulated annealing algorithm for static asset-based weapon allocation is presented in [3]. However, no evaluation of the quality of the solutions obtained by the algorithm is presented, so it is unknown how the algorithm performs.

3.2 A genetic algorithm for static weapon allocation

In [7], we present a genetic algorithm designed for real-time allocation of defensive weapon resources to targets. The original version of the algorithm was intended for the static target-based problem, but we have now with some modifications adapted it to also suit the static asset-based formulation of the problem.

The algorithm is described in pseudo code in Algorithm 1. First, an initial population consisting of $nrOfIndividuals$ is created, through generation of a vector of length $|\mathbf{W}|$. In this vector each element W_k is assigned a random integer value in the interval $\{1, \ldots, |\mathbf{T}|\}$. In each generation

Algorithm 1 Pseudo code for our genetic algorithm

```
Pop ← GenerateInitialPopulation()
while termination criteria not met do
  for l ← 1 to nrOfIndividuals do
    J_l ← CalculateFitness(Pop(l))
    if J_l > CalculateFitness(g⃗) then
      g⃗ ← Pop(l)
  Pop′ ← Crossover(Pop)
  Pop ← Mutate(Pop′)
return g⃗
```

we evaluate all individuals in the population and determine their objective function values in accordance with Equation 1. Hence, each individual is assigned a fitness value that is used in the following phases of selection and recombination. After the evaluation phase, deterministic tournament selection is used as selection mechanism to determine which individuals in population Pop that should be used as parents for Pop', i.e., we pick two individuals at random from Pop and select the one with best fitness value. When two parents have been selected from Pop, we apply one-point crossover at a randomly selected position $k \in \{1, \ldots, |\mathbf{W}|\}$, generating two individuals that become members of Pop'. This is repeated until there are $nrOfIndividuals$ in Pop'. Thereafter, we apply mutation on a randomly selected position $k \in \{1, \ldots, |\mathbf{W}|\}$ in the first individual of Pop', where the old value is changed into $i \in \{1, \ldots, |\mathbf{T}|\}$. Hence, there is a probability of $1/|\mathbf{T}|$ that the individual is unaffected of the mutation. The mutation operator is repeated on all individuals in Pop' and the resulting individuals become members of the new population Pop. This loop is repeated until the termination criterion is fulfilled (the upper limit on the computational time bound is reached). At this point, the individual with the best fitness found during all generations is returned as the solution of which weapons that should be allocated to which targets.

3.3 A particle swarm optimization algorithm for static weapon allocation

In [8], we develop a particle swarm optimization algorithm for the static target-based weapon allocation problem. We have modified this algorithm to also suit the static asset-based weapon allocation problem.

A particle swarm consists of $nrOfParticles$ particles, in which each particle is associated with a position $\vec{x_l}^t$, a velocity $\vec{v_l}^t$, and a memory $\vec{b_l}^t$ storing the particle's personal best position. Moreover, we also store the swarm's global best position in a vector $\vec{g}^t$. Each particle corresponds to a solution, given by the particle's position.

The algorithm is described in pseudo code in Algorithm 2. In an initialization phase, each particle is assigned an

Algorithm 2 Pseudo code for our particle swarm optimization algorithm

```
Initialization()
while termination criteria not met do
  for l ← 1 to nrOfParticles do
    J_l ← CalculateFitness(x⃗_l)
    if J_l = CalculateFitness(g⃗) then
      p_l ← reinitialize()
    else
      if J_l > CalculateFitness(b⃗_l) then
        b⃗_l ← x⃗_l
        if J_l > CalculateFitness(g⃗) then
          g⃗ ← x⃗_l
  for l ← 1 to nrOfParticles do
    v⃗_l ← UpdateVelocity(p⃗_l)
    x⃗_l ← UpdatePosition(p⃗_l)
return g⃗
```

initial position $\vec{x_l}^0$ (where the elements in the initial position vectors are integers randomly distributed between 1 and $|\mathbf{T}|$), and an initial velocity $\vec{v_l}^0$ (a vector of real numbers randomly distributed from the uniform distribution $U[-|\mathbf{T}|, |\mathbf{T}|]$). A fitness value is calculated for each particle, given by the objective function value J (see Equation 1) that is obtained for the solution corresponding to the particle's position. The new fitness is compared to the personal best and the global best to see whether these should be updated accordingly. After this, the velocity and position is updated for each particle, according to Equation 5 and 6.

$$\vec{v_l}^{t+1} = \omega \vec{v_l}^t + c_1 \vec{r_1}^t \circ (\vec{b_l}^t - \vec{x_l}^t) + c_2 \vec{r_2}^t \circ (\vec{g}^t - \vec{x_l}^t) \quad (5)$$

$$\vec{x_l}^{t+1} = \vec{x_l}^t + \vec{v_l}^{t+1} \quad (6)$$

In Equation 5, ω is a parameter referred to as inertia or momentum weight, specifying the importance of the previous velocity vector, while c_1 and c_2 are positive constants specifying how much a particle should be affected by the personal best and global best positions (referred to as the cognitive and social components, respectively). $\vec{r_1}^t$ and $\vec{r_2}^t$ are vectors with random numbers drawn uniformly from the interval $[0, 1]$. Moreover, the $\circ$-operator denotes the Hadamard product, i.e., element-by-element multiplication of the vectors. After the position update specified in Equation 6, we round off the particles' positions to their closest integer counterpart. In next iteration we calculate the particles'

new fitness values, whereupon the velocities and positions are updated, and so on. This is repeated until a termination criterion is met, i.e., that no more time remains. When this happens, the best solution obtained so far is returned as output from the algortihm.

A problem that must be handled is particles moving outside the bounds of the search space. When this happens, we reinitialize the position and velocity values of the coordinate for which the problem occurred. Moreover, in order to avoid premature convergence to local optima, we reinitialize the position and velocity vectors for particles rediscovering the current best solution.

3.4 A maximum marginal return algorithm for static weapon allocation

A greedy algorithm for static target-based weapon allocation, known as the maximum marginal return (MMR) algorithm, was initially suggested in [4]. This algorithm (described with pseudo code in Algorithm 3) is very simple since it works greedily by assigning weapons sequentially to the target that maximizes the reduction of the expected target value. When the first weapon has been allocated to the target for which the reduction in value is maximal, the target value is reduced to the new expected value. After that, the same procedure is repeated for the second weapon, and so on, until all weapons have been allocated to targets. It is not obvious how to use the MMR algorithm for the static asset-based weapon allocation problem, since it in this version of the problem does not exist any target values. Instead, there are protection values associated with the defended assets, and lethality values associated with the targets. In [13], it is suggested that a defended asset's weight (protection value) is equally distributed over the targets aimed for it, so that a target's value is computed as $V_i = \frac{\omega_j}{|\mathbf{G}_j|}$ (where j is the index for the set $\mathbf{G}_j$ of which target T_i is a member), and that the asset-based problem is approximated with its target-based counterpart. Similar reasoning is presented in [5] where it is suggested that the value of a target is set to the expected destroyed protection value of the defended asset to which it is aimed, given that the target is not engaged and that all other targets aimed for the defended asset are destroyed. We have here used the latter view, so that the target value V_i for a target T_i is calculated as:

$$V_i = \omega_j \times \pi_i, \tag{7}$$

where j is the index of the defended asset to which target T_i is aimed.

We have also included a variant of greedy search where we have taken the solution generated by the MMR algorithm and improved it with a simple local search that creates neighbor solutions by swapping two positions selected at random in the solution vector (this variant of the MMR algorithm, described with pseudo code in Algorithm 4, will in the following be referred to as MMR-LS). This algorithm is an implementation of the idea briefly discussed in [13].

Algorithm 3 Maximum marginal return algorithm

```
for all k such that 1 ≤ k ≤ |W| do
    highestValue ← −∞
    for all i such that 1 ≤ i ≤ |T| do
        value ← V_i × P_ik
        if value > highestValue then
            highestValue ← value
            b ← i
    greedySolution[k − 1] ← b
    V_i ← V_i × (1 − P_bk)
return greedySolution
```

Algorithm 4 The MMR-LS algorithm

```
bestSolution ← MMR()
J_best ← CalculateFitness(bestSolution)
while termination criteria not met do
    neighborSolution ← neighbor(bestSolution)
    J_new ← CalculateFitness(neighborSolution)
    if J_new > J_best then
        bestSolution ← neighborSolution
        J_best ← J_new
return bestSolution
```

Obviously, the quality of the solutions generated by the MMR-LS algorithm will always be at least as good as the quality of the solutions returned by the MMR algorithm.

4 Experiments

In the experiments reported here, we have used the open source testbed SWARD[1] (System for Weapon Allocation Research and Development). The testbed is implemented in Java, and we have been running the experiments on a computer with a 2.4GHz Intel Core Duo CPU and 3GB RAM. In an earlier version of SWARD, there was only support for experiments with algorithms for target-based weapon allocation. However, into the current version we have implemented support for asset-based weapon allocation as well. By using SWARD, we make sure that the experiments presented here are easily reproducible, so that researchers can test other algorithms on the same problem instances.

In order to recreate the problem instances used in the experiment presented in Section 4.2.1, the following settings should be used in SWARD: $T_{start} = 5$,$W_{start} = 5$, $T_{end} = 9$, $W_{end} = 9$, $T_{step} = 1$, $W_{step} = 1$, $iterations = 10$, $DAs = 5$, $seed = 0$, $TimeLimit = 1000$ms.

Similarly, for recreating the problem instances used in the experiment presented in Section 4.2.2, the following settings should be used: $T_{start} = 10$,$W_{start} = 10$, $T_{end} = 30$, $W_{end} = 30$, $T_{step} = 10$, $W_{step} = 10$, $iterations = 100$, $DAs = 5$, $seed = 0$, $TimeLimit = 5000$ms.

For the genetic algorithm we have used the parameter setting: $nrOfIndividuals = \max(|\mathbf{T}|, |\mathbf{W}|)$, while we

[1]The open source testbed SWARD can be downloaded from http://sourceforge.net/projects/sward/

for the particle swarm optimization algorithm have used $nrOfParticles = 50$, $c_1 = 2.0$, $c_2 = 2.0$, and $\omega = 0.8$.

4.1 Exhaustive search

In a first experiment, we measured the time required for performing an exhaustive search on problem instances of various size (where $|\mathbf{T}|$ and $|\mathbf{W}|$ are varied while $|\mathbf{A}|$ is fixed to 5). In order to get more robust results, we ran the exhaustive search algorithm ten times on each problem instance, and calculated an average time from these ten runs. The results are presented in Table 1. For problem sizes where $|\mathbf{T}| \leq 9$ and $|\mathbf{W}| \leq 9$ and where no time is reported, the time required is less than or equal to 1 millisecond.

Table 1: Average computation time (in milliseconds) for the exhaustive search algorithm

	$\|\mathbf{W}\|$					
$\|\mathbf{T}\|$	4	5	6	7	8	9
3			2	8	25	79
4		3	15	66	242	624
5		5	32	175	950	5061
6	2	16	104	678	4405	28160
7	4	37	284	2148	16055	119921
8	8	78	666	5782	49628	424787
9	15	146	1445	14018	134526	1293084

We can from the experimental results see that the computational time needed grows quickly. For the largest problem size tested in the first experiment, i.e., $(|\mathbf{T}| = 9, |\mathbf{W}| = 9)$, the exhaustive search algorithm on average required more than 20 minutes to return the optimal solution. This is clearly too long for all kinds of real-world air defense scenarios involving RAM, but already problem sizes demanding considerably shorter computation times may be too large for exhaustive search, taking the high speed of threatening rockets into account.

4.2 Heuristic algorithm performance

For scenarios that demand solving the weapon allocation problem faster than is possible with optimal algorithms, we have to rely on heuristic algorithms. In Section 4.2.1, we present experimental results obtained with the suggested heuristic algorithms on small-scale problem instances, while we in Section 4.2.2 present results on large-scale problem instances.

4.2.1 A comparison against the optimal solution for small-scale problems

We have in order to investigate the quality of the solutions generated by the suggested algorithms compared their obtained objective function values to the optimal objective function values obtained by the exhaustive search algorithm for relatively small-scale scenarios between $(|\mathbf{T}| = 5, |\mathbf{W}| = 5)$ and $(|\mathbf{T}| = 9, |\mathbf{W}| = 9)$.

We have calculated the offset to the optimal solution and averaged the results over ten problem instances. Each algorithm has been allowed to run for one second on each problem instance and the results are shown in Figure 1.

Looking at the offsets to the optimal solution, it can be seen that most of the algorithms create near-optimal solutions on average for small-scale problem instances. The MMR algorithm is by far the worst of the algorithms, but when allowed to improve its initial solution by local search (i.e., the MMR-LS algorithm), the quality is improved. However, we have in further experiments seen that even though the allowed search time is increased, the MMR-LS is not able to improve upon the solutions generated after one second. The genetic algorithm and the particle swarm optimization algorithm both create very good solutions, and if the search time is increased slightly, both algorithms are able to find optimal solutions for the small-scale problem instances tested here. With the tight time constraints tested, it is evident that the particle swarm optimization algorithm is the most suitable of the algorithms for small-scale problem instances.

4.2.2 A comparison between algorithms on larger-scale problems

In a second experiment with the heuristic algorithms, we have tested them on larger-scale problems ranging in between $(|\mathbf{T}| = 10, |\mathbf{W}| = 10)$ and $(|\mathbf{T}| = 30, |\mathbf{W}| = 30)$. Since it from the results presented in Section 4.2.1 is obvious that the performance of MMR-LS to a reasonable increase in computational cost improves upon the solution quality obtained by the MMR algorithm, we have in this experiment only used MMR-LS instead of both MMR and MMR-LS. The algorithms have in this experiment been allowed to run for five seconds on each problem instance. The optimal solutions are hard to obtain for large-scale problem instances, so instead of calculating the offset to the optimal solution, we have here simply plotted the objective function values obtained (averaged over 100 problem instances) in Table 2-4.

Table 2: Mean objective function values for $|\mathbf{T}| = 10$.

	10×10	10×20	10×30
GA	267.97	302.29	302.54
MMR-LS	268.80	301.19	301.92
PSO	270.77	304.16	302.89

As can be seen in Table 2, the particle swarm optimization algorithm has the highest mean objective function values for the tested scenarios consisting of ten targets, although the differences are small between the tested algorithms. The differences are larger for problem instances consisting of twenty targets, as can be seen in Table 3. As the search space increases, the quality of the solutions produced by the particle swarm optimization algorithm compared to the solutions produced by the genetic algorithm becomes worse. This can be seen in the last column of Table 3, as well as the two last columns in Table 4. These are the scenarios that have the largest search spaces (number of feasible solutions).

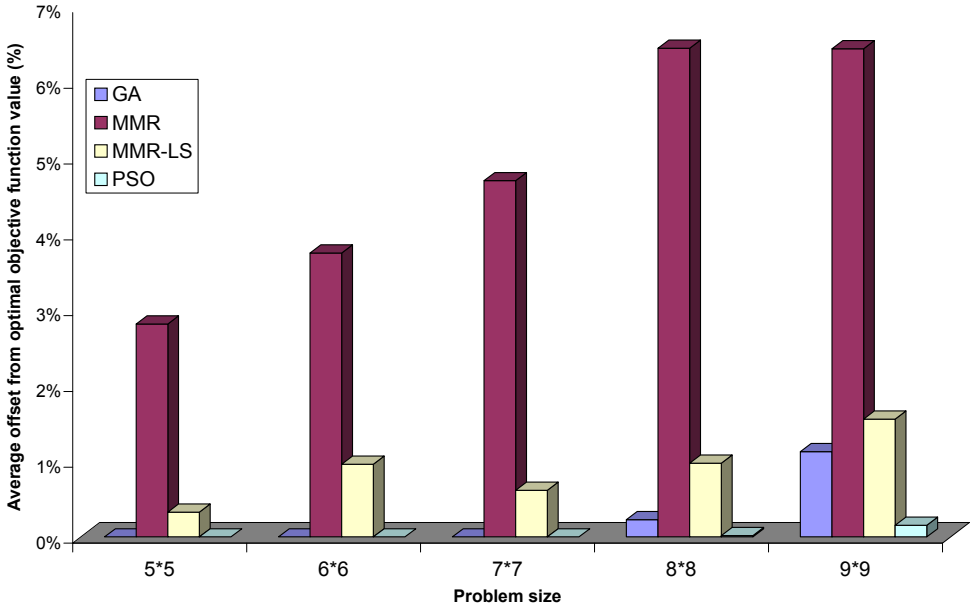

Figure 1: Offset (in %) from optimal solution for the heuristic algorithms on small-scale scenarios. Averaged over ten problem instances.

Table 3: Mean objective function values for $|\mathbf{T}| = 20$.

	20×10	20×20	20×30
GA	156.71	221.49	265.15
MMR-LS	129.31	220.26	262.98
PSO	160.47	224.07	252.83

Taking a closer look at the performance of the MMR-LS algorithm, it can be seen that it performs well (relative the other algorithms) on problem instances where the number of firing units is equal to or larger than the number of targets, but that it performs much worse on problem instances with a larger number of targets than firing units. These empirical results fit well with [5], in which it is argued analytically that the method of approximating a static asset-based problem with its target-based counterpart should give good results for problem instances with a "strong defense" but that it may give solutions of lower quality for problem instances with a "weak defense".

Table 4: Mean objective function values for $|\mathbf{T}| = 30$.

	30×10	30×20	30×30
GA	99.32	150.72	188.22
MMR-LS	60.10	125.01	176.61
PSO	103.93	144.65	163.45

Overall, the genetic algorithm perform well on all the tested problem sizes. It is clearly less good than the particle swarm optimization algorithm on smaller problem sizes, but the average offset is never more than a few percent. At the same time, it clearly outperforms the other heuristic algorithms on the problem sizes ($|\mathbf{T}| = 30, |\mathbf{W}| = 20$) and ($|\mathbf{T}| = 30, |\mathbf{W}| = 30$).

5 Conclusions and future work

We have in this paper presented the static asset-based weapon allocation problem which is an optimization problem that need to be solved in a short amount of time in air defense situations involving RAM threats such as rockets and mortars. Empirical results show that small-scale problem instances can be solved optimally very quickly, but that heuristic algorithms are needed for slightly larger problem instances. For this reason, we have implemented a genetic algorithm, a particle swarm optimization algorithm, and a greedy maximum marginal return algorithm. Such algorithms have earlier been used for the static target-based weapon allocation problem, but as far as we know, it is previously unknown how they perform on the asset-based version of the problem. Our experiments have shown that optimal or very near-optimal solutions are obtained in real-time by the genetic algorithm and the particle swarm optimization algorithm on small-scale problems. The maximum marginal return algorithm yields worse solutions, but these can easily be improved upon by local search. For larger problem instances the optimal solutions are not known, and can therefore not be used for comparison. Instead, the objective function values produced by the algorithms have been compared to each other. It has been shown that the greedy algorithm with local search creates solutions of good quality (com-

pared to the other algorithms) for scenarios with a strong defense, but that it performs bad on scenarios involving a weak defense, i.e., where there is a larger number of targets than there are firing units. The genetic algorithm performs well relative the other algorithms on all the tested problem sizes, while the particle swarm optimization algorithm seems to be better suited for small search spaces than large search spaces.

5.1 Future work

The obtained results can be used as benchmarks for other heuristic algorithms. Hence, it is our hope that the used data sets (problem instances) will be used by other researchers as well, so that a better understanding of which algorithms that work well for static asset-based weapon allocation is obtained. Moreover, in the current research on static asset-based weapon allocation, it is assumed that kill probabilities, lethality probabilities, and target aims are known with certainty. Obviously, these estimates will in real-world systems be associated with uncertainty, and it would therefore be interesting and useful to know how sensitive the solutions produced by the algorithms are to such uncertainties.

Acknowledgment

This work was supported by the Information Fusion Research Program (University of Skövde, Sweden) in partnership with Saab Electronic Defense Systems and the Swedish Knowledge Foundation under grant 2003/0104. We would like to express our thankfulness to the other members of the ground situation awareness scenario for fruitful discussions and valuable feedback.

References

[1] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization. *ACM Computing Surveys*, 35(3):268–308, 2003.

[2] Jie Chen, Bin Xin, ZhiHong Peng, LiHua Dou, and Juan Zhang. Evolutionary decision-makings for the dynamic weapon-target assignment problem. *Science in China Series F: Information Sciences*, 52(11):2006–2018, 2009.

[3] Camelia Ciobanu and Gheorghe Marin. On heuristic optimization. *An. Stiint. Univ. Ovidius Constanta*, 9(2):17–30, 2001.

[4] G. G. den Broeder, R. E. Ellison, and L. Emerling. On optimum target assignments. *Operations Research*, 7(3):322–326, 1959.

[5] Patrick A. Hosein. *A class of dynamic nonlinear resource allocation problems*. PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1990.

[6] Patrick A. Hosein and Michael Athans. Preferential defense strategies: Part 1 - the static case. Technical report, Massachusetts Institute of Technology, Laboratory for Information and Decision Systems, 1990.

[7] Fredrik Johansson and Göran Falkman. An empirical investigation of the static weapon-target allocation problem. In *Proceedings of the 3rd Skövde Workshop on Information Fusion Topics*, 2009.

[8] Fredrik Johansson and Göran Falkman. A suite of metaheuristic algorithms for static weapon-target allocation. In *Proceedings of the 2010 International Conference on Genetic and Evolutionary Methods*, 2010.

[9] Stephan E. Kolitz. Analysis of a maximum marginal return assignment algorithm. In *Proceedings of the 27th Conference on Decision and Control*, 1988.

[10] Zne Jung Lee and Wen Li Lee. A hybrid search algorithm of ant colony optimization and genetic algorithm applied to weapon-target assignment problems. In Jiming Liu, Yiu-Ming Cheung, and Hujun Yin, editors, *Proceedings of the 4th International Conference on Intelligent Data Engineering and Automated Learning*, volume 2690 of *Lecture Notes in Computer Science*, pages 278–285. Springer, 2003.

[11] S.P. Lloyd and H.S. Witsenhausen. Weapon allocation is NP-complete. In *Proceedings of the 1986 Summer Conference on Simulation*, 1986.

[12] W.P. Malcolm. On the character and complexity of certain defensive resource allocation problems. Technical Report DSTO-TR-1570, DSTO - Weapons Systems Division, 2004.

[13] William A. Metler and Fred L. Preston. A suite of weapon assignment algorithms for a SDI mid-course battle manager. Technical report, Naval Research Laboratory, 1990.

[14] Robert A. Murphey. Target-based weapon target assignment problems. In Panos M. Pardalos and Leonidas S. Pitsoulis, editors, *Nonlinear assignment problems: algorithms and applications*, pages 39–53. Kluwer Academic Publishers, 2000.

[15] Stéphane Paradis, Abder Rezak Benaskeur, Martin Oxenham, and P. Cutler. Threat evaluation and weapons allocation in network-centric warfare. In *Proceedings of the 8th International Conference on Information Fusion*, 2005.

[16] Xiangping Zeng, Yunlong Zhu, Lin Nan, Kunyuan Hu, Ben Niu, and Xiaoxian He. Solving weapon-target assignment problem using discrete particle swarm optimization. In *Proceedings of the 6th World Congress on Intelligent Control and Automation*, 2006.