Journal of the Operational Research Society (2003) 54, 958–969 

www.palgrave-journals.com/jors

# An evolutionary approach for the target allocation problem

E Erdem[1] and NE Ozdemirel[2]*

[1]*Department of Industrial Engineering, Atilim University, Ankara, Turkey; and* [2]*Industrial Engineering Department, Middle East Technical University, Ankara, Turkey*

We propose an evolutionary approach for target allocation in tactical level land combat. The purpose is to assign friendly military units to enemy units such that the total weapon effectiveness used is minimised while the attrition goals set for the enemy units are satisfied. A repair algorithm is developed to ensure feasibility with respect to the attrition goal constraints. A tightness measure is devised to determine the population size of the genetic algorithm as a function of constraint tightness. Also, a local improvement algorithm is used to further improve the solution quality. Experimental results indicate that the genetic algorithm can find solutions with acceptable quality in reasonable computation time. Although the approach is developed for the target allocation problem, it can be adapted for other assignment problems.

*Journal of the Operational Research Society* (2003) **54,** 958–969. doi:10.1057/palgrave.jors.2601580

**Keywords:** genetic algorithms; allocation; military; repair algorithms

## Introduction

Metaheuristics have frequently been used to solve various forms of assignment problems. In their extensive survey article, Osman and Laporte[1] report numerous applications proposing solution approaches for assignment problems based on constraint logic programming, evolutionary computing, neural networks, simulated annealing, tabu search, and others. When we focus on evolutionary computing, there are a number of genetic algorithms developed for solving the generalised assignment problem.[2,3] Others[4-7] tackle the quadratic assignment problem. Finally, there are genetic algorithms proposed for dealing with selection-assignment problems.[8,9]

In this study we are concerned with a special case of the assignment problem, namely the target allocation problem. The version of the target allocation problem we deal with has been defined and solved within the context of an applied research project conducted for the Turkish General Staff.[10]

The purpose in target allocation is to optimally assign heterogeneous blue (friendly) forces to heterogeneous red (enemy) forces in tactical level land combat. Forces of both sides are organised in military units. A unit chosen at the tactical level of military hierarchy consists of a small number of (usually two or three) subunits. A blue unit can be assigned as a whole to a single red unit, or it can be divided among several red units. Conversely, a red unit can be allocated one or more blue units. Hence, in general, assignment takes place in prespecified fractions (subunits) of blue units.

The objective of target allocation is to minimise total weapon effectiveness of the blue units used subject to the constraints for satisfying attrition goals set for the red units. We propose an evolutionary approach for solving the target allocation problem. In our genetic algorithm (GA), we make use of a repair algorithm (RA) to ensure feasibility of solution with respect to attrition goal constraints.

Parameters of the GA are tuned using problems of three relatively small sizes and three constraint tightness levels. Based on these initial results, we have devised a tightness measure that can be used, together with the problem size, in choosing the population size of the GA. We have also developed a local improvement algorithm to further increase the solution quality. Finally, we have experimented with the GA extensively to estimate its solution quality and computation time for larger problems.

Although originally intended for the target allocation problem, our solution approach can easily be generalised to assignment problems where resources can be divided in predetermined (discrete) portions among demand points.

The target allocation problem is defined in the next section, followed by descriptions of the GA and the RA. Then, experimental results and additional development including the tightness measure and the local improvement algorithm are discussed. Finally, conclusions and further research topics are given.

*Correspondence: NE Ozdemirel, Industrial Engineering Department, Middle East Technical University, Ankara 06531, Turkey.
E-mail: nurevin@ie.metu.edu.tr

## Problem definition

We assume that the two opposing sides involved in the battle are composed of heterogeneous forces, that is, different types of units. A unit, however, is homogeneous in itself. For example, an infantry unit may consist of 100 infantry soldiers, whereas there may be six tanks in a tank unit. We deal with land combat at tactical level as opposed to strategic or operational levels, hence we consider brigades or smaller units.

Considering force division and combination effects with fractional assignments is our contribution in extending target allocation models reported in the literature.[11-13] Our assumptions concerning force division and combination are as follows.

A1. A blue unit can divide its force among at most three red units. Considering that a military unit typically consists of two or three subunits division between two red units can be with fractions (1/3, 2/3) or (1/2, 1/2) for protecting subunit integrity. In case of division among three red units, the fractions are (1/3, 1/3, 1/3).

A2. When a blue unit divides its force among multiple red units, its total attrition potential will be lower compared to the case when it is assigned as a whole to a single red unit. This is called the division effect.

A3. When multiple blue units are assigned to a single red unit, their total attrition potential will be higher than the mere sum of individual attrition potentials due to synergistic effects. This is called the combination effect.

A4. If a portion of a blue unit is to be used, then the entire unit will be assigned. This assumption can be relaxed to allow partial assignments. Division and combination effects are assumed to be present with partial assignments as well.

When a blue unit is divided, its integrity is lost, command and control activities are disrupted, certain valuable weapon systems that are few in number become unavailable to some subunits, logistic support may get more difficult, and some combat scenarios cannot be implemented any longer. Division effect defined in assumption A2 reflects loss in attrition power due to such reasons. Assumption A3 implies that, when units are combined, synergy and increase in the variety of human skills and weapon systems results in gain in attrition power. Interestingly, the combination effect may also take the form of loss in case of overkills, that is, a blue unit wastes its power by firing at a red weapon system that has already been destroyed by another blue unit.

Some allocation combinations may not be meaningful when assigning blue units to red units. For example, the attrition an infantry unit can give to an artillery unit located at a distance is virtually none, or some allocations may not be possible due to terrain conditions. For this reason, we give the set of possible allocations, represented by an allocation graph, as input to the model. The allocation graph also includes the division codes indicating possible division alternatives for blue units. An example of such a graph is given in Figure 1(a). The battle takes place between six blue and six red infantry units. Blue unit $i3$ has division code 24 and, according to Table 1, this means that it can be assigned to red units $j3$ and $j4$ either with fractions (1/3, 2/3) or (2/3, 1/3). A sample solution for this allocation graph is illustrated in Figure 1(b) when assumption A4 is relaxed. Note that unassigned portions of all blue units are assigned to a virtual red unit, $j0$, which will later be used in the RA. The complete set of assignment alternatives when assumption A4 is relaxed are given in Table 1 for each division code.

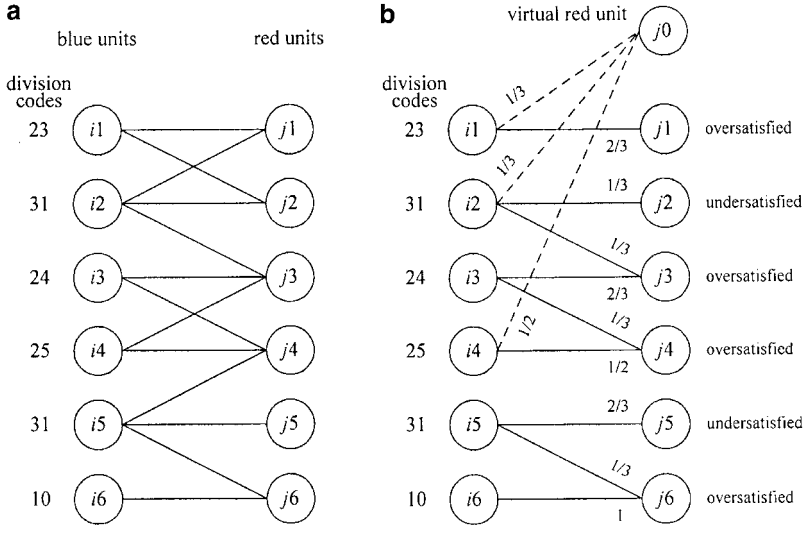


**Figure 1** An instance of the target allocation problem: (a) allocation graph; (b) a sample solution.

**Table 1** Division codes and assignment alternatives when assumption A4 is relaxed

| *Division code* | *Division alternatives* | *Assignment alternatives* |
|---|---|---|
| 10 | No division | (0), (1) |
| 20 | No division | (0, 1) (1, 0) (0, 0) |
| 21 | (1/2, 1/2) | (0, 1/2) (1/2, 0) (1/2, 1/2)<br>(0, 1) (1, 0) (0, 0) |
| 22 | (1/3, 2/3) | (0, 2/3) (1/3, 0) (1/3, 2/3)<br>(0, 1) (1, 0) (0, 0) |
| 23 | (2/3, 1/3) | (0, 1/3) (2/3, 0) (2/3, 1/3)<br>(0, 1) (1, 0) (0, 0) |
| 24 | (1/3, 2/3) or (2/3, 1/3) | (0, 2/3) (1/3, 0) (1/3, 2/3)<br>(0, 1/3) (2/3, 0) (2/3, 1/3)<br>(0, 1) (1, 0) (0, 0) |
| 25 | (1/2, 1/2) or (1/3, 2/3) | (0, 1/2) (1/2, 0) (1/2, 1/2)<br>(0, 2/3) (1/3, 0) (1/3, 2/3)<br>(0, 1) (1, 0) (0, 0) |
| 26 | (1/2, 1/2) or (2/3, 1/3) | (0, 1/2) (1/2, 0) (1/2, 1/2)<br>(0, 1/3) (2/3, 0) (2/3, 1/3)<br>(0, 1) (1, 0) (0, 0) |
| 27 | (1/2, 1/2) or (1/3, 2/3) or (2/3, 1/3) | (0, 1/2) (1/2, 0) (1/2, 1/2)<br>(0, 2/3) (1/3, 0) (1/3, 2/3)<br>(0, 1/3) (2/3, 0) (2/3, 1/3)<br>(0, 1) (1, 0) (0, 0) |
| 30 | No division | (0, 0, 1) (0, 1, 0) (1, 0, 0) (0, 0, 0) |
| 31 | (1/3, 1/3, 1/3) | (0, 0, 1/3) (0, 1/3, 0) (1/3, 0, 0)<br>(0, 1/3, 1/3) (1/3, 0, 1/3) (1/3, 1/3, 0)<br>(0, 0, 2/3) (0, 2/3, 0) (2/3, 0, 0)<br>(0, 1/3, 2/3) (1/3, 0, 2/3) (1/3, 2/3, 0)<br>(0, 2/3, 1/3) (2/3, 0, 1/3) (2/3, 1/3, 0)<br>(1/3, 1/3, 1/3)<br>(0, 0, 1) (0, 1, 0) (1, 0, 0) (0, 0, 0) |

We now give the notation used in the target allocation model and its formulation.

- $i$ blue unit index, $i=1,\ldots, I$
- $j$ red unit index, $j=1,\ldots, J$
- $x_{ij}$ decision variable indicating fraction of blue unit $i$ assigned to red unit $j$
- $WEI_i$ weapon effectiveness index (value or cost) of blue unit $i$
- $\alpha_j$ attrition goal for red unit $j$ to be reached within the target battle duration
- $b'_{ij}$ total net attrition given by blue unit $i$ to red unit $j$ *after* division and combination effects are considered

$$\min .\ z = \sum_{i=1}^{I}\sum_{j=1}^{J} WEI_i x_{ij} \tag{1}$$

$$\text{s.t.} \sum_{i=1}^{I} b'_{ij} x_{ij} \geq \alpha_j, \quad j = 1,\ldots,J \tag{2}$$

$$\sum_{j=1}^{J} x_{ij} \leq 1, \quad i = 1,\ldots,I \tag{3}$$

$$x_{ij} \in \{0, 1/3, 1/2, 2/3, 1\}, \quad i = 1,\ldots,I, \quad j = 1,\ldots,J$$

The above is a simplified conceptual formulation where objective (1) is to minimise the total value or cost of blue forces used subject to constraints (2) for satisfying red attrition goals. According to constraint set (3), when assumption A4 is relaxed, a blue unit can be unused, used totally, or used with a fraction as dictated by the allocation graph. The target allocation model is in fact an integer programming model where binary decision variables are used to indicate force allocations. The real formulation[10] is nonlinear in attrition goal constraints because modelling synergistic effects requires multiplication of decision variables. The constraints are linearised by using additional binary indicator variables for detecting divisions and combinations. Moreover, there are some technical constraints to regulate unit inclusion/exclusion in combinations and divisions. These indicator variables and technical constraints increase the problem size significantly.

The problem is known to be NP-hard. We have been able to solve small instances by total enumeration, but larger instances where $I, J \geq 10$ cannot be solved by conventional integer programming techniques within desired time frame. As an alternative to enumeration, we have developed a solution approach based on decomposition (by blue units) and column generation. This approach has failed to produce allocations in allowable discrete fractions. The repair and

improvement heuristics applied to the column generation solution have yielded much worse solution quality than the GA even for small instances.

Among the model parameters, weapon effectiveness index of a unit can be found by commercially available systems such as TASCFORM[14] as a function of the weapon systems and ammunition available in the unit as well as the terrain and geographical conditions. Attrition goal for red unit $j$ is found as $\alpha_j = \mu_j R_{j0}$, where $0 \leq \mu \leq 1$ is the target fraction of loss in the unit to be reached within the target battle duration and $R_{j0}$ is the initial force level of the unit. Target battle duration and $\mu_j$ are specified by the decision maker as part of the battle strategy.

In finding $b'_{ij}$ coefficients, we first estimate $b_{ij}$ by running a Lanchester simulation model one-to-one for each $(i, j)$ pair. (Lanchester models[15,16] are basically systems of differential equations that can be used for deterministic continuous simulation of combat.) If the remaining force level at the end of the target battle duration is found as $R_j$ by Lanchester simulation, then $b_{ij} = R_{j0} - R_j$ is the attrition given by blue unit $i$ to red unit $j$ in the battle *before* division and combination effects.

We define the division and combination effect parameters as follows:

$\lambda(i, k_i)$ Loss in attrition potential of blue unit $i$ when it is divided among $k_i$ red units, $k_i = 1, 2, 3$, $\lambda(i, 1) = 0$, $0 \leq \lambda(i,2) \leq \lambda(i, 3) \leq 1$.

$\phi(i, j, l_j)$ Gain in attrition potential of blue unit $i$ against red unit $j$ when $l_j$ blue units are combined against red unit $j$, $l_j = 1, 2, \ldots$, $\phi(i, j, 1) = 0$, $0 \leq \phi(i, j, 2) \leq \phi(i, j, 3) \leq 1$, $\phi(i, j, l_j) = \phi(i, j, 3)$ for $l_j > 3$.

Then, the coefficients needed in constraint set (2) can be expressed as $b'_{ij} = (1 - \lambda(i, k_i))(1 + \phi(i, j, l_j))b_{ij}$. Values of division and combination parameters are to be determined by military experts. These values reflect their aspiration levels regarding the loss or gain in attrition potential. In case of overkills, it is possible to use as the combination effects the overkill rates that are estimated from war game simulations or military exercises.

## Genetic algorithm

The decisions that should be made in developing a GA can be summarised as follows:[17]

- choosing a representation scheme for the solution,
- defining the fitness function that will be used,
- deciding how to create the initial population,
- defining the genetic operators that will be used for parent selection, crossover, mutation, and replacement,
- defining the termination rule,
- choosing the GA parameters such as population size and mutation probability.

We discuss each one of these issues in this section except the last one, which will be considered in the experimentation section. In addition, we need to deal with the feasibility issue and decide how to handle the attrition goal constraints.

### *Representation scheme and fitness function*

The representation scheme we have devised for the target allocation problem is somewhat unconventional. In a chromosome, we have a gene for every blue unit, which is a vector whose size is defined by the first digit of the unit's division code. Each member of this vector is a pair that indicates the red unit index to which the blue unit is assigned and the allocation fraction. The chromosome representation of the sample solution given in Figure 1(b) can be written as

$$(i1 \rightarrow j1 : 2/3,\ j2 : 0)$$
$$(i2 \rightarrow j1 : 0,\ j2 : 1/3,\ j3 : 1/3)$$
$$(i3 \rightarrow j3 : 2/3,\ j4 : 1/3)$$
$$(i4 \rightarrow j3 : 0,\ j4 : 1/2)$$
$$(i5 \rightarrow j4 : 0,\ j5 : 2/3,\ j6 : 1/3)$$
$$(i6 \rightarrow j6 : 1)$$

Choice of the fitness function comes naturally as the objective function of the model given in Equation 1.

### *Initial population generation*

Initial population is generated randomly with the help of the *generation tables*. A generation table is prepared for each division code in accordance with the assignment alternatives listed for that division code in Table 1. Given the division code of a blue unit in the allocation graph, a gene is generated for the unit by randomly selecting one of the assignment alternatives from the respective generation table. Genes generated for all blue units are then combined to form a chromosome.

This generation scheme ensures that the chromosomes are in accordance with the allocation graph. However, there is no guarantee that the attrition goal constraints given in inequality (2) are satisfied. Several methods are suggested for constraint handling in GAs. These are[18] (a) using problem specific representation schemes and genetic operators, (b) repairing infeasible chromosomes, (c) penalising infeasible chromosomes in the fitness function, and (d) rejecting infeasible chromosomes.

We use a problem specific representation scheme and special genetic operators to ensure feasibility with respect to the allocation graph. For the attrition goals, however, we adapt the repair strategy for creating a feasible initial population. Liepins *et al*[19] have shown, through empirical test of GAs on a diverse set of constrained combinatorial optimisation problems, that the repair strategy surpasses other strategies in both speed and solution quality. Repair strategy has been used successfully by many researchers.[20–22]

The RA, which will be described in the next section, is called for every infeasible chromosome in an attempt to convert it to a feasible one. If the repair attempt fails, the chromosome is discarded and a new one is generated from scratch. This process is repeated until either a feasible population of size $N$ is obtained or the desired population size cannot be achieved after $NN \gg N$ chromosomes are generated. When the latter occurs, the algorithm stops by reporting the best feasible solution (if any) or reports that the problem instance is infeasible. Otherwise, given a population of $N$ feasible chromosomes, fitness values are computed and the best one is recorded as the incumbent solution. The GA then moves to parent selection for reproduction.

### *Parent selection, crossover, and mutation*

For parent selection, the fitness values are transformed by using the normalisation scheme suggested by Cheng and Gen,[23] that is,

$$f' = \frac{f_{\max} - f + \gamma}{f_{\max} - f_{\min} + \gamma}$$

where $f'$ is the transformed fitness, $f$ is the original fitness, $f_{\max}$ and $f_{\min}$ are the maximum and the minimum fitness values in the population, and $0 < \gamma < 1$ is a transformation constant mainly used to avoid division by zero.

We use two-point crossover, which is found to perform slightly better than one-point crossover and uniform crossover.[24,25] When the number of crossovers per generation, $C$, is one, the first parent is the incumbent solution. The second parent is selected randomly where the selection probabilities are proportional to the transformed fitness values. The two crossover points are selected at random, and the genes between these points are interchanged between the parents to obtain two offspring. For $C > 1$, the first parent in the first crossover is again the best solution. The others are selected according to their fitness proportional probabilities, implying roulette wheel selection.

Each offspring is mutated with the mutation probability $p_m$. Mutation involves changing a randomly selected gene of a chromosome. Generation tables created for initial population generation are also used for mutation. The selected gene is replaced by a randomly selected assignment alternative from the relevant generation table.

Note that offspring generated by our crossover and mutation operators conform with the allocation graph. However, their feasibility with respect to attrition goal constraints is not guaranteed as in the case of initial population generation.

### *Feasibility check, repair, and replacement*

Feasibility check is performed by computing the *amount of undersatisfaction*. $B_j$ is the difference between two sides of each attrition goal constraint in an offspring, that is, $B_j = \alpha_j - \sum_{i=1}^{I} b'_{ij} x_{ij}$. If $B_j > 0$, then the respective constraint is violated or red unit $j$ is undersatisfied, otherwise it is (over)satisfied. Oversatisfied chromosomes are directly kept under consideration as candidates for the next generation.

We employ two alternative replacement strategies for infeasible chromosomes. In the first one ($RS = 1$), we call the RA in an attempt to repair the offspring and keep it under consideration if the repair is successful. Otherwise, we still keep the unrepaired chromosome as a candidate if

$$\frac{\sum_{j \in SI} B_j}{\sum_{j=1}^{J} \alpha_j} \leq IFL \text{ and } f \leq \bar{f} - SDM\sigma$$

where $SI$ is the index set of undersatisfied red units, $0 < IFL < 1$ is an infeasibility limit, $\bar{f}$ and $\sigma$ are average and standard deviation of fitness values in the population, and $SDM$ is the standard deviation multiplier, which is typically taken as one, two, or three. This means that solutions that are infeasible within a certain limit will still be considered, provided that they have fitness values significantly better than the population average.

In the second replacement scheme ($RS = 2$), we again call the RA to repair an infeasible offspring. If the repair is successful, we replace the original chromosome with the repaired one with a replacement probability, $p_r$. With probability $1-p_r$, we keep the original chromosome but take as its fitness value the one found for the repaired version. If the repair is unsuccessful, then we discard the chromosome.

In forming the population for the next generation, we simply sort all existing chromosomes in the current generation and the offspring that are kept under consideration in increasing fitness. Then we choose the best $N$ of them as the new population. The incumbent solution is updated at this point if necessary.

Once the new population is formed, the generation counter is incremented and the algorithm starts over with parent selection. This cycle continues until the generation counter reaches a prespecified generation limit, $G$. The algorithm then stops by reporting the incumbent solution as the solution to the problem instance.

## Repair algorithm

The RA is called at two points in the GA to convert infeasible solutions to feasible ones, when creating a feasible initial population and after applying mutation to an offspring. The main idea behind the RA is to switch portions of blue units from oversatisfied red units to undersatisfied ones. If this is not sufficient to eliminate the infeasibility, then the algorithm starts assigning unused portions of blue units to undersatisfied red units. The RA chooses units for switching in a greedy manner using certain criteria. We explain these criteria on an example using the following notation:

$SI$ index set of undersatisfied red units in the solution to be repaired

$e$ index of the undersatisfied red unit currently under consideration which the repair algorithm will attempt to satisfy, $e \in SI$

$SA$ index set of blue units that can be assigned to red unit $e$ according to the allocation graph

$SF$ index set of oversatisfied red units for which $x_{ij} > 0$ and $i \in SA$

$d$ index of the oversatisfied red unit currently under consideration from which the RA will attempt to switch some portion of blue units to red unit $e$, $d \in SF$

$SD$ index set of blue units for which $x_{id} > 0$ and the division code $> 10$

$SS$ index set of blue units that are assigned to oversatisfied red unit $d$ and that can be switched to undersatisfied red unit $e$, $SS = SA \cap SD$

$c$ index of the blue unit currently under consideration some portion of which will be switched from red unit $d$ to red unit $e$, $c \in SS$

Suppose that the sample solution given in Figure 1(b) represents a solution obtained when generating the initial population or after applying mutation to an offspring. Assume that $j2$ and $j5$ are undersatisfied in this solution as $B_j > 0$ for $j = 2, 5$, hence $SI = \{j2, j5\}$. We choose as $e$ the most undersatisfied red unit, that is, $e = \arg\max_{j \in SI}\{B_j/\alpha_j\}$. Supposing $B_{j2}/\alpha_{j2} = 0.2$ and $B_{j5}/\alpha_{j5} = 0.1$, $e = j2$. Then $SA = \{i1, i2\}$ and $SF = \{j1, j3\}$, which means that we can consider switching some fractions of $i1$ and $i2$ from $j1$ and $j3$ to $j2$ in an attempt to satisfy $j2$. In doing this, however, we do not allow $j1$ or $j3$ to become undersatisfied.

We consider as $d$ the most oversatisfied red unit, that is, $d = \arg\min_{j \in SF}\{B_j/\alpha_j\}$. If $B_{j1}/\alpha_{j1} = -0.15$ and $B_{j3}/\alpha_{j3} = -0.3$, then $d = j3$. Then $SD = \{i2, i3\}$, $SS = \{i2\}$, and $c = i2$. In the presence of multiple blue candidates for switching, we choose the one offering the largest relative gain in attrition potential, that is, $c = \arg\max_{i \in SS}\{b_{ie}/\alpha_e - b_{id}\alpha_d\}$.

Once the units involved in switching are selected, the RA makes use of the *switch tables*. An example of a switch table is given in Table 2 for division code 27. When the RA considers switching some portion of blue unit $c$ from oversatisfied red unit $d$ to undersatisfied red unit $e$, the table is traversed from top to bottom until the row matching the current allocation of $c$ is found. Then the algorithm tries to switch according to the portions in the subsequent rows, gradually increasing the portion to be switched until either $e$ is satisfied, $d$ is undersatisfied, or portion allocated to $d$ becomes zero, whichever occurs first. Note that, in this table, only those rows having zero in the third column will be used under assumption A4.

**Table 2** Switch table for division code 27

| *Fraction of c assigned to d* | *Fraction of c assigned to e* | *Fraction of c assigned to others* |
|---|---|---|
| 1 | 0 | 0 |
| 2/3 | 1/3 | 0 |
| 1/2 | 1/2 | 0 |
| 1/3 | 2/3 | 0 |
| 0 | 1 | 0 |
| 2/3 | 0 | 1/3 |
| 0 | 2/3 | 1/3 |
| 1/2 | 0 | 1/2 |
| 0 | 1/2 | 1/2 |
| 1/3 | 0 | 2/3 |
| 0 | 1/3 | 2/3 |

If red unit $e$ is still undersatisfied after the first switching, other candidates are considered; first the next $c \in SS$ for the same $d$ and then the next $d \in SF$ are tried until all switching possibilities are exhausted. Remaining $e \in SI$ are also tried to be satisfied by means of switching.

If there are still some undersatisfied red units after switching, then the RA starts assigning unused portions of blue units. Creating a virtual red unit $j0$ with $\alpha_{j0} = 0$ and assuming all unused blue fractions are assigned to this virtual red unit allows us to use a similar switching logic in this second phase of the algorithm. There are two main differences in this case. Firstly, $d = j0$ and the blue unit to be assigned to undersatisfied red unit $e$ is selected with the criterion $c = \arg\min_{i \in SE}\{WEI_i/b_{ie}\}$, where $SE$ is the index set of blue units that have unused portions that can be assigned to red unit $e$. Secondly, the fitness value needs to be updated by adding the cost of additional blue units used.

If all red units are satisfied at the end of the second phase, the RA returns the repaired chromosome indicating feasibility. Otherwise, it returns the partially repaired or unrepaired chromosome and indicates that the repair attempt has failed.

## Experimentation and additional development

We have experimented with the GA in several stages. Firstly, we have conducted some pilot runs to decide on which parameters of the algorithm we need to experiment with. In the second stage, using small randomly generated problems, we have tuned the GA parameters by a factorial experiment. Then, based on the same results, we have decided that there is a need for a tightness measure to be used in choosing the population size and there is room for local improvement. In the last stage, we have experimented with the final version of the GA on larger problems to evaluate its performance in terms of solution quality and computation speed.

### *Pilot runs*

Defining the size of the target allocation problem instance as $I \times J$, we have used 10 problems in the pilot runs whose sizes vary between $6 \times 5$ and $24 \times 24$. Some of these problems are based on the scenarios developed in the applied research project, and some are obtained by combining smaller problems. Partly looking at the solution quality of these

problems and partly using suggestions from the literature, we have fixed some parameters of the GA and decided to experiment further with others.

Considering suggestions from the literature[25,26], and our problem sizes, we have set the population size as $N = 30$. The number of trials for generating a feasible initial population is chosen as $NN = 500\,000$ since the task may require many trials for problems with tight constraints. We have experimented with 500 and 1000 as the generation limit and, by examining convergence behaviour of the GA, found that $G = 500$ is sufficient. We have tried 0.1 and 0.9 for the transformation constant $\gamma$ and, seeing that it does not make a difference, fixed it at 0.4.

We have decided to further experiment with the number of crossovers per generation and set levels for this parameter as $C = 1$, 5 and 10. Similarly, we have selected $p_m = 0.3$ and 0.4 as experimental levels for the mutation probability. Values suggested in literature for $p_m$ are 0.001 and 0.01 per bit when binary representation is used.[25,26] In our case, only one gene in the entire chromosome is mutated with $p_m$, hence we have chosen much larger values.

Of the two replacement strategies, $RS = 1$ is applied either with $IFL = 0$ (no infeasible solutions are allowed in the population) or with $IFL = 0.05$ and $SDM = 2$. The parameter values in the latter case are found to be too restrictive in permitting unrepaired chromosomes to enter the population, hence we have decided to set $IFL = 0.10$ and $SDM = 1$ in further runs. $RS = 2$, on the other hand, is implemented with $p_r = 0.05$ as suggested by Orvosh and Davis.[22] Although we have tried different $p_r$ values, $RS = 2$ is found to be inferior to $RS = 1$ and eliminated from further consideration.

### *Test problem generation*

Our test problems vary in size from $6 \times 6$ to $96 \times 96$. We have generated the smallest problems from scratch and the larger problems by combining randomly selected small ones. For example, $12 \times 12$ and $24 \times 24$ problems are obtained by combining two and four $6 \times 6$ problems, respectively. The reason for this is that we can find the optimal solution for a small problem by simple enumeration of all assignment alternatives given in the allocation graph. Optimal solution for a larger problem is then found as the sum of the optimal solutions of small problems comprising the larger problem. This way we can compare the GA's solution quality with the optimum.

In generating a $6 \times 6$ problem, we first come up with a viable allocation graph. We randomly generate the division code for each blue unit, assuming all 11 division codes are equally likely. Given the division code, we select the appropriate number of red units to which a blue unit can be assigned. We assume that, under the terrain restrictions, blue unit $i1$ can be assigned to red units $j1$–$j3$, $i2$ to $j1$–$j4$, $i3$ and $i4$ to $j2$–$j5$, $i5$ to $j3$–$j6$, and $i6$ to $j4$–$j6$. After red units are selected at random, we check if all red units are covered to avoid infeasibility. Also, if a blue unit with a division code 20 or 30 is assigned to two or three red units and no other blue units are assigned to the same red units, we again have a coverage problem since the blue unit cannot be divided. In such cases, we discard the problem and generate a new one from scratch.

The next step is generation of the problem parameters. Weapon effectiveness indices are generated uniformly between 500 and 1000 for the blue units and between 200 and 500 for the red units, assuming the red side is defending and needs less power. Then the attrition coefficients are found as $b_{ij} = WEI_i/WEI_j$. Note that $b_{ij}$ can take values between one and five. Division effects $\lambda(i, k_i)$ are uniform between 0.08 and 0.13, whereas combination effects $\phi(i, j, l_j)$ are between 0.05 and 0.10.

Attrition goals $\alpha_j$ are generated uniformly between one and five in accordance with the $b_{ij}$ values. Then a final check is performed with the criterion

$$\sum_{i \in SK_i} b_{ij} \geq 2\alpha_j, \quad j = 1, \ldots, 6$$

where $SK_j$ is the index set of blue units that can be assigned to red unit $j$ according to the allocation graph. Here $\alpha_j$ is multiplied by two since on the average only half of the blue units in $SK_j$ can be assigned to a red unit. This check is necessary for increasing the chances that the problem instance has a feasible solution. If this condition is not satisfied, the problem is discarded and a new one is generated.

From a generated problem of any size, we obtain three problems, each at a different level of tightness in attrition goal constraints. A relaxed problem is obtained by multiplying all $\alpha_j$ values generated for that problem by 0.3, moderate problems by 0.6, and tight problems by 0.9. Hence we have 15 problem categories all together (five problem sizes $\times$ three tightness levels) and generated 50 problems in each category.

### *Tuning the GA parameters*

We have carried out a factorial experiment using problems of size up to $24 \times 24$ for tuning the three GA parameters, namely $C$, $p_m$, and $IFL$. The results are summarised in Table 3, where each figure in the table is the average over 50 problems of percent deviation of the GA solution from the optimal solution. Analysis of variance indicates that $C = 1$ and $p_m = 0.4$ yield better solution quality in general. $IFL$ does not make a significant difference, hence we choose $IFL = 0$.

In Table 3 we observe that the solution quality deteriorates as the problem size increases and as the problems become more relaxed in terms of attrition goal

**Table 3** Results of the factorial experiment

| | | *Average % deviation from optimum with parameter values* | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *C = 1* | | | | *C = 5* | | | | *C = 10* | | | |
| *Problem characteristic* | | *IFL = 0* | | *IFL = 0.1* | | *IFL = 0* | | *IFL = 0.1* | | *IFL = 0* | | *IFL = 0.1* | |
| *Tightness level* | *Problem size* | $p_m = 0.3$ | $p_m = 0.4$ | $p_m = 0.3$ | $p_m = 0.4$ | $p_m = 0.3$ | $p_m = 0.4$ | $p_m = 0.3$ | $p_m = 0.4$ | $p_m = 0.3$ | $p_m = 0.4$ | $p_m = 0.3$ | $p_m = 0.4$ |
| Relaxed | 6 × 6 | 3.62 | 4.78 | 3.61 | 4.76 | 4.94 | 5.64 | 4.93 | 5.65 | 5.10 | 5.10 | 5.11 | 5.09 |
| | 12 × 12 | 6.28 | 5.81 | 6.27 | 5.81 | 6.75 | 6.26 | 6.76 | 6.27 | 7.20 | 7.32 | 7.20 | 7.31 |
| | 24 × 24 | 7.34 | 6.66 | 7.33 | 6.67 | 6.92 | 7.20 | 6.91 | 7.20 | 6.87 | 6.88 | 6.88 | 6.87 |
| Moderate | 6 × 6 | 2.34 | 1.87 | 2.34 | 1.89 | 2.39 | 2.28 | 2.38 | 2.27 | 2.46 | 2.45 | 2.46 | 2.46 |
| | 12 × 12 | 2.93 | 3.30 | 2.93 | 3.31 | 3.59 | 3.58 | 3.59 | 3.57 | 3.50 | 3.57 | 3.51 | 3.58 |
| | 24 × 24 | 4.79 | 4.91 | 4.79 | 4.90 | 5.15 | 5.07 | 5.15 | 5.06 | 5.34 | 5.29 | 5.35 | 5.30 |
| Tight | 6 × 6 | 1.03 | 1.10 | 1.02 | 1.11 | 1.13 | 1.16 | 1.13 | 1.16 | 1.16 | 1.16 | 1.16 | 1.16 |
| | 12 × 12 | 1.29 | 1.21 | 1.29 | 1.21 | 1.54 | 1.45 | 1.53 | 1.45 | 1.54 | 1.43 | 1.54 | 1.44 |
| | 24 × 24 | 1.51 | 1.44 | 1.50 | 1.45 | 1.66 | 1.69 | 1.66 | 1.69 | 1.77 | 1.76 | 1.77 | 1.76 |

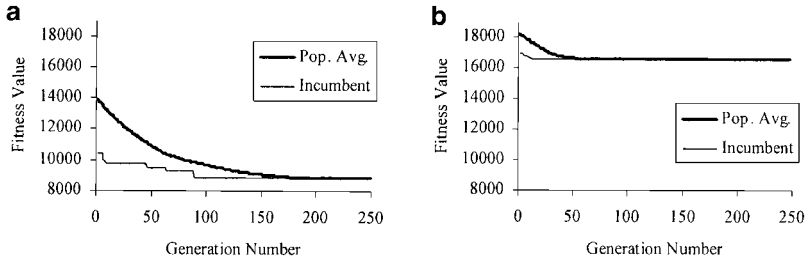


**Figure 2** Convergence behaviour of the GA: (a) relaxed version of a 24 × 24 problem; (b) tight version of a 24 × 24 problem.

constraints. This is expected because the search space gets larger as the problems get larger and more relaxed.

When we examine convergence behaviour of the algorithm for relaxed *versus* tight problems, we discover an additional effect due to the RA. As an example, convergence of the GA is depicted in Figure 2 for relaxed and tight versions of a 24 × 24 problem where $C = 1$, $p_m = 0.4$, and $IFL = 0$. In relaxed problems, there is not as much need for repair as in tight problems, and the GA itself is more effective in finding the solution. In tight problems, however, it is the RA that finds the solution rather than the GA. The RA is deterministic; therefore, repair attempts result in similar chromosomes, reducing diversity of the population. Nevertheless, the greedy RA seems to be an effective algorithm by itself in finding good quality feasible solutions.

### *Choosing the population size based on the problem size and tightness*

Considering the effect of the problem size and tightness on the solution quality, we have thought that defining the population size as a function of these two problem characteristics might be a good idea. In fact, Robertson[27] reports that the performance increases as the population size increases. Goldberg[28] proposes increasing the population size as the problem size increases.

We have devised a tightness measure to be used, in conjunction with the problem size, in choosing the population size for a problem instance. In this measure, we express the tightness of individual constraints by

$$R_j = \frac{\alpha_j}{(I/J) \times (\sum_{i \in SK_j} b_{ij} / |SK_j|)}$$

where $SK_j$ is the index set of blue units that can be assigned to red unit $j$ according to the allocation graph. The denominator is roughly the expected attrition power that can be allocated to red unit $j$. Then we find the overall problem tightness, $TM$, as the average of all $R_j$ values. Note that as the value of $TM$ gets closer to zero, the problem becomes more relaxed.

We then choose the population size as $N = \max\{30, I, \lceil 30/\mathrm{TM} \rceil\}$. Here, we do not allow $N$ to be smaller than 30 and increase it further either according to the problem size or the problem tightness, whichever yields a larger value. In case the above expression results in too large a population size because $I$ is extremely large or $TM$ is extremely small, an upper limit may also be specified.

*Local improvement algorithm*

In an attempt to further improve the solution quality especially for relaxed problems, we have developed a local improvement algorithm (LI). (Solving relaxed problems is particularly important because in many attack scenarios blue-to-red force ratio is three-to-one, and the problem is relaxed.) Upon mutation, the GA calls the RA for each infeasible offspring and the LI for each feasible offspring. The LI tries to reduce the fitness value of an offspring by decreasing blue unit fractions assigned to red units without violating attrition goal constraints. In doing this, we first choose a blue unit, $u$, and then a red unit, $v$, as

$$u = \arg\max_{i} \left\{ WEI_i \Big/ \sum_{i=1}^{I} WEI_i \right\},$$
$$v = \arg\max_{j \in SU_u} \left\{ \frac{B_j}{b_{uj}\alpha_j} \Big/ \sum_{j=1}^{J} \frac{B_j}{b_{uj}\alpha_j} \right\} \quad (4)$$

where $SU_u$ is the index set of red units to which blue unit $u$ is assigned in the solution. Hence, we first choose the blue unit having the largest contribution to the fitness value. Then, in making a choice among the relevant red units, we consider the relative slacks in attrition goal constraints of the red units as well as the contribution of the chosen blue unit to attrition of red units. We then try to reduce allocation fractions of blue unit $u$ starting with red unit $v$. (We have also tried selecting first $v$ and then $u$ among the blue units that are assigned to $v$ and observed that this yields inferior solution quality.)

In order to keep the population diversity intact and to avoid premature convergence, we have also devised a stochastic version of the LI. In this version, $u$ and $v$ are selected randomly where the selection probabilities are found by the expressions in curly brackets in Equation 4.

*Experimental results*

We have experimented with the final version of the GA where the population size is determined by the problem size and tightness. We have solved 50 problems in each of the 15 problem categories without the LI and with deterministic (DLI) and stochastic (SLI) versions of the LI. Problem characteristics and computation times are given in Table 4 as the average of 50 problems in each category. After trying $N = 70{,}100$, and 130, we have set the upper limit on the population size as 100 since the marginal contribution of further increase is found to be insignificant. The number of optimal solutions, and average and maximum percent deviation from the optimum in each case are given in Table 5. Of the $6 \times 6$ problems generated, one moderate and 10 tight problems are found to be infeasible and not used in generating larger problems.

According to Table 4, the limiting population size is needed for relaxed problems. For moderate and tight problems, the GA uses population sizes larger than 30, which are determined mostly by the tightness measure. When we compare the GA solution quality of the three smallest size problems with constant population size (for parameters $C = 1$, $p_m = 0.4$, and $IFL = 0$ in Table 3) and with

**Table 4** Problem characteristics and computation time (average of 50 problems)

| *Problem Characteristic* | | | | *Computation time (s)** | | | |
|---|---|---|---|---|---|---|---|
| *Tightness level* | *Problem size* | *Tightness measure, TM* | *Population size, N* | *Initial pop. generation time* | *GA evolution time* | *GA + DLI evolution time* | *GA + SLI evolution time* |
| Relaxed | $6 \times 6$ | 0.23 | 100.00 | 0.77 | 1.04 | 1.04 | 1.05 |
| | $12 \times 12$ | 0.23 | 100.00 | 1.12 | 1.52 | 1.54 | 1.56 |
| | $24 \times 24$ | 0.23 | 100.00 | 1.18 | 2.64 | 2.80 | 3.04 |
| | $48 \times 48$ | 0.23 | 100.00 | 9.76 | 6.24 | 6.48 | 7.36 |
| | $96 \times 96$ | 0.23 | 100.00 | 435.88 | 15.02 | 17.48 | 21.35 |
| Moderate | $6 \times 6$ | 0.46 | 67.78 | 1.04 | 0.65 | 0.69 | 0.83 |
| | $12 \times 12$ | 0.43 | 70.12 | 0.85 | 1.52 | 1.58 | 1.61 |
| | $24 \times 24$ | 0.43 | 69.68 | 1.68 | 2.88 | 2.94 | 3.08 |
| | $48 \times 48$ | 0.44 | 69.26 | 2432.92 | 6.42 | 6.84 | 7.01 |
| | $96 \times 96$ | 0.44 | 96.00 | 36 383.00† | 13.35 | 15.42 | 17.43 |
| Tight | $6 \times 6$ | 0.68 | 45.34 | 5.64 | 0.80 | 0.84 | 0.86 |
| | $12 \times 12$ | 0.65 | 46.92 | 0.98 | 1.62 | 1.74 | 1.78 |
| | $24 \times 24$ | 0.65 | 46.56 | 81.56 | 2.72 | 2.85 | 3.07 |
| | $48 \times 48$ | 0.66 | 48.14 | 12 473.00‡ | 6.74 | 6.94 | 7.03 |
| | $96 \times 96$ | 0.65 | 96.00 | 96 819.00§ | 14.53 | 15.74 | 16.62 |

*The algorithm is coded in C and runs on a Pentium III 733 MHz PC.
†$NN = 1\,000\,000$ trials.
‡$NN = 750\,000$ trials.
§$NN = 1\,500\,000$ trials.

**Table 5** Number of optimal solutions and deviation from optimum

| *Problem characteristic* | | *Number of optimal solutions* | | | *Average % deviation from optimum* | | | *Maximum % deviation from optimum* | | |
|---|---|---|---|---|---|---|---|---|---|---|
| *Tightness level* | *Problem size* | *GA* | *GA + DLI* | *GA + SLI* | *GA* | *GA + DLI* | *GA + SLI* | *GA* | *GA + DLI* | *GA + SLI* |
| Relaxed | $6 \times 6$ | 32/50 | 47/50* | 47/50* | 1.77 | 0.57 | 0.24* | 15.44 | 10.71 | 0.98* |
| | $12 \times 12$ | 12/50 | 35/50* | 30/50 | 3.09 | 0.80 | 0.39* | 14.66 | 8.77 | 3.08* |
| | $24 \times 24$ | 3/50 | 7/50* | 5/50 | 5.60 | 1.75 | 1.25* | 17.24 | 6.52 | 5.15* |
| | $48 \times 48$ | 0/50 | 3/50* | 0/50 | 9.22 | 2.54* | 2.91 | 20.04 | 8.15 | 7.97* |
| | $96 \times 96$ | 0/50 | 0/50 | 0/50 | 15.51 | 4.84* | 5.39 | 22.72 | 8.70* | 9.93 |
| Moderate | $6 \times 6$ | 33/49 | 41/49* | 41/49* | 1.33 | 0.69 | 0.47* | 8.83 | 8.30* | 8.30* |
| | $12 \times 12$ | 17/50 | 27/50* | 23/50 | 2.11 | 1.07 | 0.65* | 7.44 | 4.26* | 4.26* |
| | $24 \times 24$ | 0/50 | 6/50* | 2/50 | 3.89 | 1.73 | 1.70* | 10.02 | 5.63* | 7.15 |
| | $48 \times 48$ | 0/50 | 1/50* | 0/50 | 6.53 | 4.30 | 4.16* | 21.96 | 21.96* | 21.96* |
| | $96 \times 96$ | 0/50 | 0/50 | 0/50 | 9.18 | 8.79 | 8.76* | 13.48 | 13.48* | 13.48* |
| Tight | $6 \times 6$ | 36/40 | 38/40* | 37/40 | 0.41 | 0.17* | 0.21 | 6.96 | 5.18* | 5.18* |
| | $12 \times 12$ | 30/50 | 41/50* | 38/50 | 0.90 | 0.34 | 0.32* | 3.97 | 4.00 | 3.48* |
| | $24 \times 24$ | 7/50 | 14/50* | 8/50 | 1.33 | 0.86* | 0.94 | 5.50 | 4.98* | 4.98* |
| | $48 \times 48$ | 0/50 | 0/50 | 0/50 | 4.80 | 4.08 | 4.04* | 9.08 | 9.08* | 9.08* |
| | $96 \times 96$ | 0/50 | 0/50 | 0/50 | 6.77 | 6.62 | 6.54* | 10.80 | 10.80 | 10.68* |

*Best of three versions for the problem category.

variable population size (Table 5), we see that average percent deviation has reduced by 1.06–3.01, 0.54–1.19, and 0.11–0.69% for relaxed, moderate, and tight problems, respectively. We can conclude that using larger population sizes determined by a tightness measure consistently improves the solution quality, especially for relaxed problems.

When we examine the computation times given in Table 4, we see that GA evolution times with or without the LI are reasonable compared to an average of 243.38 s required to find the optimal solution for a $6 \times 6$ problem by enumeration. The time-consuming task seems to be generating a feasible initial population for tight problems. Note that we have increased *NN* further to achieve the desired population size for larger and tighter problems. This, however, may not be necessary in practice, and we may use a smaller population size to reduce the computation time. As expected, all times increase as problem size gets larger, but problem tightness does not have a significant effect on evolution and local improvement times when variable population size is used.

According to Table 5, the stand-alone GA consistently reaches a better solution quality in tight problems than in relaxed problems. When either version of LI is used in GA, however, the largest average percent deviation values are obtained in moderate problems. In all tightness levels, solution quality deteriorates as the problem size increases. This is expected as the size of the search space increases exponentially in the problem size whereas the population size remains almost constant and is limited by 100.

Addition of both LI versions brings significant improvement over stand-alone GA in all problem categories. For example, without the LI, the average percent deviation for $96 \times 96$ problems is 15.51, 9.18, and 6.77% with relaxed, moderate, and tight problem versions. These deviations are reduced to 4.84, 8.79, and 6.62% with DLI. Similar reductions are observed with SLI. DLI and SLI are most effective with relaxed problems, but they do not make as much difference in tight problems. This is natural since there is not much room left for improvement in tighter problems to begin with.

When we compare DLI and SLI in terms of the number of optimal solutions. DLI dominates SLI in all problem categories. When we examine the average percent deviation figures, however, the choice between DLI and SLI is not clear, but SLI seems to perform slightly better than DLI. Maximum percent deviation figures also indicate that SLI is better than DLI for relaxed problems.

## Conclusion

We have developed a GA for the target allocation problem, an RA for satisfying attrition goal constraints, a tightness measure for the problem, and an LI. The GA population size is determined as a function of the problem size and tightness. The RA is used for repairing infeasible chromosomes, whereas the LI is called for improving feasible ones in a greedy manner.

Our experiments show that a single crossover per generation (ie steady-state evolution) and a mutation

probability of 0.4 give better solution quality. These choices slowdown the convergence, balancing the deterministic RA's effect of reducing population diversity and giving the GA a better chance for exploring the search space especially in relaxed problems. Also, variable population size determined depending on the problem size and tightness consistently gives better solution quality than constant population size, especially for relaxed problems.

Computation times required for evolution and local improvement are acceptable, although generation of a feasible initial population may take a long time for tight problems. All times increase as problem size gets larger, but problem tightness does not have a significant effect on evolution and local improvement times.

The GA alone consistently yields a better solution quality in tight problems than in relaxed problems. After addition of the LI, however, moderate problems become more difficult to solve. In all tightness levels, solution quality deteriorates as the problem size increases. This is reasonable since the size of the search space increases much faster than the population size as the problem size increases. However, the rate of deterioration is much slower (almost linear for tight problems) compared to the rate of expansion in the search space.

Both deterministic and stochastic versions of LI bring significant improvement in solution quality at a very low additional computation cost. DLI performs better than SLI in terms of the number of optimal solutions found. The LI shows similar behaviour in average percent deviation from optimal solution.

The promising results obtained by variable population size encourages us to develop procedures for choosing other GA parameters as a function of problem characteristics. For example, the mutation probability can be larger for tight problems to delay convergence. The parameters can also be changed adaptively during the evolution.

Initial population can be generated in a different manner, perhaps by using a simple heuristic rather than randomly, in an attempt to reduce the computation time and to improve the final solution quality. In doing this, providing sufficient diversity in the initial population would be a major concern. Also, more effective ways can be found for allowing infeasible but fit chromosomes to enter the population. One alternative would be using a penalty function for infeasible chromosomes rather than trying to repair them.

One may consider developing a stochastic version of the RA, employing a similar approach as in SLI. However, it should be remembered that this may occasionally cause the repair attempt to be unsuccessful. Moreover, as both the RA and the LI are found to be very effective in solving the problem, a stand-alone heuristic can be developed by combining the two together.

*Acknowledgments*—This research is funded by the General Staff of Turkish Armed Forces under the contract #00-0307-2-01-05.

## References

1 Osman IH and Laporte G (1996). Metaheuristics: a bibliography. *Ann Oper Res* **63**: 513–623.
2 Chu PC and Beasley JE (1997). A genetic algorithm for the generalised assignment problem. *Comput. Oper Res* **24**: 17–23.
3 Wilson JM (1997). A genetic algorithm for the generalised assignment problem. *J Opl Res Soc* **48**: 804–809.
4 Ahuja RK, Orlin JB and Tiwari A (2000). A greedy genetic algorithm for the quadratic assignment problem. *Compu Oper Res* **27**: 917–934.
5 Gong D, Yamazaki G, Gen M and Xu W (1999). A genetic algorithm method for one-dimensional machine location problem. *Int J Prod Econ* **60-61**: 337–342.
6 Hwang H and Sun J (1996). A genetic-algorithm-based heuristic for the GT cell formation problem. *Comput Indus Eng* **30**: 941–955.
7 Tate DM and Smith AE (1995). A genetic approach to the quadratic assignment problem. *Comput Oper Res* **22**: 73–83.
8 Herrera F, Lopez E, Mendana C and Rodriguez MA (1999). Solving an assignment-selection problem with verbal information and using genetic algorithms. *Eur J Opl Res* **119**: 326–337.
9 Zhao L, Tsujimura Y and Gen M (1996). Genetic algorithm for robot selection and work station assignment problem. *Comput Indus Eng* **31**: 599–602.
10 Özdemirel NE and Kandiller L (2001). *Semi-dynamic Modeling of Heterogeneous Land Combat*., Technical Report 01-11, Industrial Engineering Department, Middle East Technical University: Ankara, Turkey.
11 Jaiswal NK (1997). *Military Operations Research; Quantitative Decision Making*. Kluwer: Dordrecht, Netherlands.
12 Ng KYK and Lam MN (1995). Force deployment in a conventional theatre-level military engagement. *J Opl Res Soc* **46**: 1063–1072.
13 Shephard RW *et al* (1988). *Applied Operations Research: Examples from Defense Assessment*. Plenum Press: New York.
14 Regan JM and Wogt WJ (1988). *The TASCFORM Methodology: A Technique for Assessing Comparative Force Modernization*, 3rd edn. Technical Report TR-5192-1-2 Analytic Sciences Corporation: Reading, MA.
15 Taylor J (1973). Target selection in Lanchester combat: linear-law attrition process. *Naval Res Logistics Quart* **20**: 673–697.
16 Taylor J (1974). Lanchester-type models of warfare and optimal control. *Naval Res Logistics Quart* **21**: 79–106.
17 Gen M and Cheng R (1997). *Genetic Algorithms and Engineering Design*. Wiley: New York.
18 Michalewicz Z and Fogel DB (2000). *How to Solve It: Modern Heuristics*. Springer: Berlin.
19 Liepins G, Hilliard M, Richardson J and Pallmer M (1990). Genetic algorithm applications to set covering and travelling salesman problems. In: Brown D (ed). *OR/AI: The Integration of Problem Solving Strategies 1990*. Kluwer: New York, pp 29–53.
20 Liepins G and Potter W (1991). A genetic algorithm approach to multiple fault diagnosis. In: Davis LD (eds) *Handbook of Genetic Algorithms 1991*. Van Nostrand Reinhold: New York, pp 237–250.
21 Nakano R and Yamada T (1991). Conventional genetic algorithms for job-shop problems. In: Belew RK and Booker LB (eds) *Proceedings of the Fourth International Conference on*

*Genetic Algorithms*. Morgan Kaufmann: San Mateo, CA, pp 477–479.
22 Orvosh D and Davis L (1994). Using a genetic algorithm to optimize problems with feasibility constraints. In: Michalewicz Z *et al* (eds). *Proceedings of the First IEEE Conference on Evolutionary Computation*. IEEE Press: Orlando, FL, pp 548–552.
23 Cheng R and Gen M (1994). Evolution program for resource constrained project scheduling problem. In: Michalewicz Z *et al* (eds). *Proceedings of the First IEEE Conference on Evolutionary Computation*. IEEE Press: Orlando, FL, pp 736–741.
24 De Jong KA and Spears WM (1990). An analysis of the interacting roles of population size and crossover in genetic algorithms. In: Schwefel HP and Manner R (eds). *Parallel Problem Solving from Nature, First Workshop*. Springer-Verlag: Dortmund, Germany, pp 38–47.
25 Schaffer JD, Caruana RA, Eshelman LJ and Das R (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. In: Schaffer JD (ed). *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann: San Mateo, CA, pp. 51–60.
26 Grefenstette JJ (1986). Optimization of control parameters for genetic algorithms. *IEEE Trans Systems, Man Cybernet* **16**: 122–128.
27 Robertson GG (1988). Population size in classifier systems. In: Laird J (ed). *Proceedings of the Fifth International Conference on Machine Learning*. Morgan Kaufmann: Los Altos CA, pp. 142–152.
28 Goldberg DE (1985). *Optimal Initial Population Size for Binary Coded Genetic Algorithms*, TCGA Report Number 850001, The Clearinghouse for Genetic Algorithms. Department of Engineering Mechanics, University of Alabama.

*Received August 2002*
*accepted February 2003 after one revision*