

# Dynamic Strike Force Asset Allocation using Genetic Search and Case-Based Reasoning

**Sushil J. Louis**

Genetic Algorithm Systems Laboratory  
Department of Computer Science  
University of Nevada, Reno  
sushil@cs.unr.edu

**John McDonnell, Nick Gizzi**

Space and Naval Warfare Systems Center  
53560 Hull Street  
San Diego, CA 92152  
{mcdonn, gizzi}@spawar.navy.mil

## Abstract

This paper presents a new approach to the problem of allocating strike force assets in a dynamic targeting environment. We develop a nonlinear programming formulation that encompasses both strike and suppression responsibilities as well as multi-target and multi-threat allocations. Our approach uses a genetic algorithm augmented with a case-based memory, containing population members from past problem solving attempts, to obtain better performance over time on sequences of similar allocation problems. The case-base acts as an associative long-term memory of problem solving experience. Rather than starting with a randomly initialized population on each new allocation problem, we periodically inject a genetic algorithm's population with *appropriate cases (encoded allocation strategies)* from similar, previously solved problems. Using hamming distance as a simple distance (similarity) metric for choosing appropriate cases, our experimental results demonstrate the performance gains from our approach and show that our system learns to take less time to provide quality solutions to new allocation problems as it gains experience from solving other similar allocation problems.

**Keywords:** Asset Allocation, Genetic Algorithms, Associative Memory, Case-based Reasoning

## 1 Introduction

We use a system that improves its performance with experience to attack the dynamic real-time targeting and retargeting problem. The problem is to allocate a set of platforms to targets in order to maximize a non-linear objective function. Genetic algorithms were designed to deal with poorly understood non-linear problems, but tend to require large numbers

of objective function evaluations to offer viable solutions. We thus augment the genetic algorithm with a learning component to speed up the combined system and provide high quality solutions in a real-time dynamic environment.

Genetic algorithms (GAs) are randomized parallel search algorithms that search from a population of points [6, 3, 2]. Current genetic algorithm based machine learning systems use rules to store past experience to improve their performance over time [6, 3, 15, 7, 4]. However, many application areas, are more suited to a case-based storage of past experience. We propose and describe a system that uses a case-base as a long term knowledge store in a new, genetic-algorithm-based learning system that learns from experience. Results from applying this combined system to the strike force allocation problem show that our system, with experience, takes less time to solve new problems and produces better quality solutions.

Typically, a genetic algorithm randomly initializes its starting population so that it can proceed from an unbiased sample of the search space. However, problems do not usually exist in isolation and we often confront sets of similar problems. It makes sense to start a new problem solving attempt based on information from previous problem solving attempts that may have yielded useful information about the search space. Our combined system implements this by periodically injecting a genetic algorithm's population with *relevant* (we describe what we mean by relevant later) solutions or partial solutions to similar previously solved problems. If we refer to these partial solutions as cases, this kind of case injection, can provide information (a search bias) that reduces the time taken to find a quality solution to the current problem. Our approach borrows ideas from case-based reasoning (CBR) in which old problem and solution

information, stored as cases in a case-base, helps solve a new problem [14]. In our system, the data-base, or case-base, of problems and their solutions supplies the genetic problem solver with a long term memory. The system does not require a case-base to start with and can bootstrap itself by learning new cases from the genetic algorithm’s attempts at solving a problem.

The case-base does what it is best at – memory organization; the genetic algorithm handles what it is best at – adaptation. The resulting combination takes advantage of both paradigms; the genetic algorithm component delivers robustness and adaptive learning while the case-based component speeds up the system. The **Case Injected Genetic Algorithm** (CIGAR) system presented in this paper can be applied in a number of domains from computational science and engineering to operations research [9, 11, 10]. We concentrate on the strike force allocation problem in this paper.

We formulate the allocation problem in the next section, then describe the CIGAR system. Subsequently, we delineate the difference between problem and solution similarity. In the results section, we apply our system to a specially constructed sequence of allocation problems. The last section presents conclusions and directions for future work.

## 2 Strike Force Asset Allocation

The essence of strike force planning consists of allocating a collection of strike assets to a set of targets. An air strike package typically consists of attack, fighter support, suppression of enemy air defenses (SEAD), and  $C^2$  elements.

The dynamic nature of the battlefield environment poses a particularly challenging problem for mission strike force operations. Dynamic reallocation of assets may be warranted when threats “pop-up” during the course of tactical air operations or inclement weather sets in. Other conditions that may also warrant dynamic reallocation of strike force assets include compromised or disabled assets, unexpended ordnance, and the realization of higher priority targets.

Many investigations have been conducted into the optimization of strike force assets. Most, if not all, of these consider the problem of allocating assets to targets prior to launch. For example, Griggs [5] formulated a mixed-integer program (MIP) to allocate platforms and ordnance for each objective. Their MIP is augmented with a decision tree that determines the best plan based upon weather data. Li [8]

converts a nonlinear programming formulation into a MIP problem. Although risk is not explicit in this formulation, it does account for defender suppression. Finally, Yost [16] provides a survey of work conducted that addresses the optimization of strike allocation assets.

A genetic algorithm has also been used for determining optimal aircraft to target allocations. Abrahams [1] formulates a nonlinear objective function that is used for evaluating allocation strategies. While accounting for effectiveness, joint effectiveness, and risk, their formulation constrains the number of platforms to a single objective. This is unrealistic in light of the fact that a SEAD asset may act to suppress several threats at any given time. Similarly, an attack asset may be expected to attack multiple targets on a sortie.

We assume that weaponry load-outs have been predetermined based upon targeting priorities. Once a strike package is in theater the enemy’s state may have changed, significantly effecting the preplanned mission. Instead of aborting the mission, the strike force assets can be reassigned to new targeting objectives assuming reallocation occurs in a timely manner. This problem allocates strike force assets in a dynamic targeting environment.

The air-strike package consists of a variety of platforms, each of which has a specific role in the mission. A platform’s effectiveness depends on the assets loaded and proficiency of the pilot. The mathematical formulation of the problem is based on the work done by Abrahams [1]. Modifications were made to accommodate pilot (or smart munitions) performance capabilities and relax the single platform-to-objective constraint that had been imposed.

The goal of the optimizer is to select the most effective platform(s)  $P_1, \dots, P_m$  to achieve the targeting objectives  $T_1, \dots, T_n$ . The platform allocation strategy is represented as an  $m \times n$  matrix  $X$  where  $x_{ij} = 1$  if platform  $P_i$  is assigned to objective  $T_j$  and 0 otherwise. A platform can be assigned to more than one objective up to a maximum of  $M_i$  that a platform  $P_i$  can accommodate.

Each platform has a load-out that is configured prior to launch. The total number of assets in the overall strike package is represented as a set of assets  $A = A_1, \dots, A_q$ . The binary variable  $\Gamma_{ij}$  represents the assignment of the  $j$ th asset to the  $i$ th platform. Once the platforms are allocated to targeting objectives, the optimizer assigns the most effective weaponry from the existing load-out to achieve the desired effectiveness. This assignment of resources may be represented as a binary  $n \times q$  matrix  $\Omega$  that

indicates asset  $A_j$  has been allocated to objective  $T_i$ .

The proficiency of the pilot of platform  $P_i$  in using on-board asset  $A_j$  is represented as  $\delta_{ij}$ . This term can also accommodate the usage of smart munitions that have characteristics independent of a particular pilot's capabilities. If  $w_{ij}$  represents the effectiveness (probability) of asset  $A_j$  achieving objective  $T_i$ , then, assuming statistical independence between on-board assets in achieving the targeting objectives, the probability of platform  $P_i$  achieving objective  $T_k$  is

$$p_{ik} = 1 - \prod_{j=1}^q (1 - \Gamma_{ij} \delta_{ij} \Omega_{kj} w_{jk})$$

where  $q$  is the total number of assets in the strike package.

The overall probability of an objective  $T_k$  being achieved by all platforms in the strike package is given by

$$s_k = 1 - \prod_{i=1}^m (1 - x_{ik} p_{ik})$$

where  $m$  is the number of platforms. We also incorporate a priority term,  $\rho_i$ , for each objective  $T_i$  and include it in the objective function as

$$U(X) = \sum_{j=1}^n \rho_j s_j$$

where  $n$  is the number of objectives.

The coupling of assets to jointly achieve an objective can provide a marginal benefit as described by

$$V(X) = \sum_{j=1}^n \sum_{k1=1}^q \sum_{k2=1}^q \Phi_{jk1k2} \Omega_{jk1} \Omega_{jk2}$$

where  $\Phi_{jk1k2}$  is the joint effectiveness added by simultaneously assigning assets  $A_{k1}$  and  $A_{k2}$  to the same objective,  $T_j$ . It is important to note that  $\Phi$  can be negative as well as positive.

The risk to the platforms, as well as their value, should also be addressed in the objective function. If each platform  $P_i$  has an associated value  $v_i$ , and the risk associated with allocating platform  $P_i$  to objective  $T_j$  is  $r_{ij}$ , then the cost term

$$Y(X) = \sum_{i=1}^m \sum_{j=1}^n r_{ij} x_{ij}$$

needs to be included in our objective function.

The resulting objective function provides a measure of effectiveness of the strike as well as an evaluation of the risks and costs entailed in carrying out the

mission. Combining the effectiveness terms,  $U(X)$  and  $V(X)$ , with the risk/cost component,  $Y(X)$ , yields the function to be optimized

$$J(X) = \alpha U(X) + \beta V(X) + \gamma Y(X) \quad (1)$$

$J(X)$  should be maximized in order to maximize the effectiveness while minimizing the risk. In our current study, we set  $\alpha, \beta, \gamma$  to 1.0.

As stated earlier, the dynamic nature of the problem arises from the fluid nature of the battlespace. Various unforeseen factors may require a quick re-allocation of assets (re-targeting). Genetic algorithms usually require many objective function evaluations to come up with a promising solution. This implies that a genetic algorithm may be unable to provide a viable solution within stringent timelines. The combined genetic algorithm case-based reasoning system described in the next section learns from experience (off-line or on-line) and reduces the time needed to obtain a quality solution. This system offers anytime behavior and provides a promising alternate approach.

### 3 Case Injected Genetic Algorithm

We describe one way of combining a genetic algorithm with a case-based memory. When confronted with a (new) problem, we **periodically inject** a small number of previously saved solutions/chromosomes *similar to the current best member* of the GA population into the *current* population, replacing the worst members. We call this the "closest to the best" strategy, one of several possible injection strategies. The rest of the population is initialized randomly and the GA continues searching with this combined population. This system is shown in figure 1 assumes that similar solutions must have come from similar problems and avoids the issue of defining problem similarity metrics<sup>1</sup>.

#### 3.1 Cases

What are cases and how do we save them to the case-base? To answer these questions consider every individual in the population generated by the GA as a potential case for a subsequent similar problem. Unlike the typical CBR problem of finding enough cases from which to adapt, we have too many potential

<sup>1</sup>That similar solutions come from similar problems is admittedly a strong assumption but it seems to work on the problems that we tried

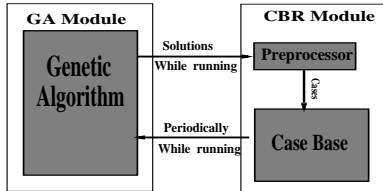


Figure 1: Conceptual view of CIGAR.

cases. For example, a population of size 100 run for 100 generations generates 10000 potential cases from just one problem solving attempt. We therefore use the following strategy. During GA search, whenever the fitness of the best individual in the population increases, the new best individual is stored in the case-base. A case is a member of the population (a candidate solution) together with ancillary information including fitness, the generation this case was generated, and its parents [12]. Reusing old solutions has been a traditional performance improvement procedure. Our work differs in that 1) we attack a set of tasks, 2) store and reuse **intermediate** candidate solutions, and 3) do not depend on the existence of a problem similarity metric avoiding indexing problems common to case-based systems.

What happens if our similarity measure is noisy and/or leads to unsuitable retrieved solutions? By definition, unsuitable solutions will have low fitness and will quickly be eliminated from the GA’s population. CIGAR may suffer from a slight performance hit but will not break or fail – the genetic search component will continue making progress toward a solution. CIGAR is robust. As noted earlier, we can choose schemes other than injecting the closest to the best; furthest from the worst and probabilistic versions of closest to the best and furthest from the worst have also proven effective.

There is much work in machine learning that is relevant to this paper. We elide this discussion in the interest of saving space and refer the interested reader to [9].

What do we mean by problem similarity? What is the difference between problem similarity and solution similarity? The next two sections provide concrete answers to these questions.

## 4 Indexing and Similarity

Indexing, or how we define similarity, is a basic issue in all the systems described above. Previous work has dealt with the simpler case where a similarity metric exists in the problem space [10, 13]. Simply

put, when we know that two problems are similar, the system can use information from attempting one problem to improve performance on the other. However, a problem similarity metric is not easy to come by and remains a major issue for case-based reasoning systems. In this paper, we study what we believe is the more realistic case where we do *not* need (or may not have) a similarity measure on the problem space. Genetic algorithms solutions are encoded as strings and our results show that purely syntactic similarity measures on the solution space lead to surprisingly good performance – a needed property for applications to poorly-understood systems. Specifically, since we are using a binary string to encode a candidate solution, **hamming distance** serves as our similarity metric.

Each individual was encoded in a binary string composed from substrings corresponding to each platform (see Figure 2). Each substring encodes the tar-

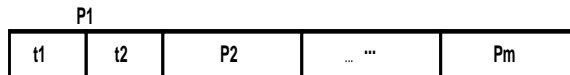


Figure 2: An individual represents an allocation of platforms to targets

gets that the platform has been allocated and the concatenation of these platform substrings encodes the allocation of platforms to targets for the strike force.

## 5 Implementation

The problem that we constructed had ten platforms, ten targets, and forty assets. Each platform could be allocated to 2 targets. We need four (4) bits to associate a unique id for each of 10 targets and thus each platform’s substrings consists of  $4 \times 2 = 8$  bits and for 10 targets we end up with a chromosome length of  $8 \times 10 = 80$  bits.

### 5.1 Problem Generation

In this preliminary work, we first construct a strike force allocation problem,  $A_0$ , by using fixed values for all platform, pilot, target, and asset properties. For example, all elements of the proficiency matrix were set to 1.0. Risk was the only factor that varied. We generated a risk matrix by initializing the diagonal elements of the matrix to 0.0 and all other elements to 1.0. Thus the optimal allocation for this initial problem was platform  $P_i$  to target  $T_i$ . We generate  $A_1$  by randomly choosing two rows or two columns

and swapping them and repeat this process  $(mn)^{0.5}$  times. This ensures that there is exactly one 0.0 entry in every row and column and all other entries are 1.0, thus keeping the problem simple. Continuing in this way, using  $A_1$  as a basis for  $A_2$ , and  $A_2$  as a basis for  $A_3$  and so on, we generate 50 problems  $A_0$  through  $A_{49}$ . Note that problems close to each other in the sequence are probabilistically more similar than problems farther apart in the sequence.

We used the three-operator genetic algorithm described in Goldberg’s book [3] but modified the selection operator to be elitist. In our selection strategy, if the population size is  $N$  then, the offspring produced by crossover and mutation initially double the population. The new generation consists of the best  $N$  individuals from the combined  $2N$  parents and offspring.

Our selection strategy induces strong convergence and needs to be balanced by high crossover and mutation rates. We have found that two point crossover with crossover rate of 1.0 coupled with point mutations with a mutation rate of 0.05 works well and these were used in our reported results. A population size of 80 was used and the genetic algorithm ran for 80 iterations.

The case-base in this first prototype is simply a flat text file of encoded solutions (chromosomes). Before CIGAR starts on a new problem, it reads the case-base into an array in memory. Our similarity measure is hamming distance between the chromosomes.

## 6 Results

We compare the performance of a Randomly Initialized Genetic Algorithm (RIGA) and CIGAR over these 50 problems. Both algorithms use exactly the same genetic algorithm parameters and all results reported are averages over 10 runs. We used a population of 80, and ran the algorithms for 80 generations. We obtain the same qualitative results, as described below, with other parameter choices.

CIGAR used the “probabilistic closest to the best” strategy for case injection in the following results and we injected 10% of the population with individuals from the case base every  $(\text{population size})^{0.5} = 9$  generations [9]. The injection interval corresponds to the time needed for an individual to take over the population under the influence of our elitist selection.

The graphs in the next section are averages over ten independent runs with the parameters defined above. Figure 3 compares the time to convergence

of a GA versus CIGAR while Figure 4 compares the quality of the converged solution produced by a GA versus that produced by CIGAR. Specifically, Figure 3 plots average time taken to find the best solution on the vertical axis and the number of problems solved on the horizontal axis. Figure 4 plots average objective function value on the vertical axis and the number of problems solved on the horizontal axis. The fairly straight dashed lines on both

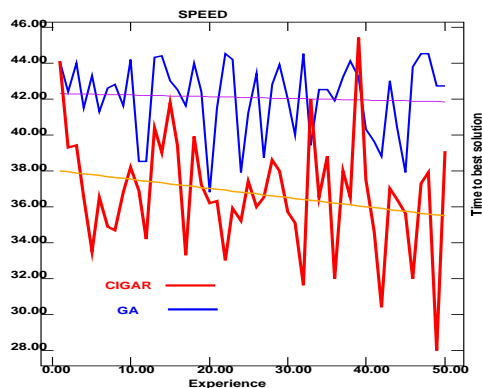


Figure 3: Time to best solutions: genetic algorithm (GA) versus Case Injected Genetic Algorithm (CIGAR)

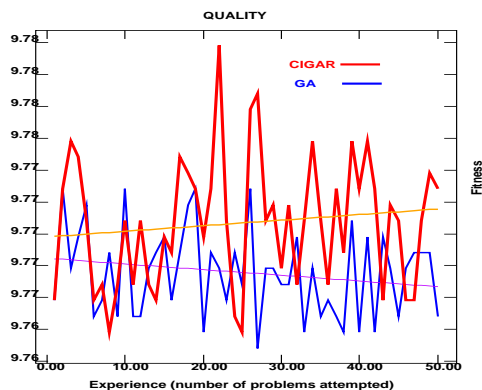


Figure 4: Solution quality: genetic algorithm (GA) versus Case Injected Genetic Algorithm (CIGAR)

graphs are the linear regression lines for the distributions represented by our performance metrics, time to convergence and quality of solution at convergence. These regression lines provide a simple model to predict performance trends. The figures clearly show that CIGAR takes less and less time to produce better and better quality solutions as it gains experience from attempting more problems. We have similar results from using other injection strategies and other

sizes of problems, but do not have the space to show these results.

## 7 Discussion and Conclusions

This paper offers a new mathematical formulation of the strike package asset allocation problem and shows that a genetic algorithm offers a viable optimization algorithm for this problem. To cater to stringent real-time constraints we described a learning system that combines genetic algorithms with case-based reasoning. The Case Injected GA (CIGAR) makes the assumption that similar problems have similar solutions. According to the schema theorem, genetic algorithms process syntactic string similarities and we have shown that storing and injecting syntactically similar solutions leads to increased performance with experience. CIGAR thus combines the strengths of GAs and CBR and avoids the difficult problem of measuring similarities among problems.

Our preliminary results indicate that, in the context of asset allocation, CIGAR learns to increase performance (both objective function value and time to convergence) at related tasks as it gains experience. We show that the simple hamming distance syntactic similarity measure works well. Our hypothesis is that we can train the combined system through simulations (off-line) so that the system learns to quickly provide high quality alternative allocation strategies (on-line) to a decision maker. We next plan to study the effect of changes in platform, target, asset, and pilot properties, singly and in combination, as well as behavior of the system under different system-parameter choices.

### Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 9624130.

## References

- [1] P. Abrahams, R. Balart, J. S. Byrnes, D. Cochran, M. J. Larkin, W. Moran, G. Ostheimer, and A. Pollington. Maap: the military aircraft allocation planner. In *Evolutionary Computation Proceedings of the IEEE World Congress on Computational Intelligence*, pages 336–341. IEEE press, 1998.
- [2] David B. Fogel. *Evolutionary Computation, Toward a New Philosophy of Machine Intelligence*. IEEE Press, New York, NY, 1995.
- [3] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
- [4] J. Grefenstette, C. Ramsey, and A. Shultz. Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5:355–381, 1990.
- [5] B. J. Griggs, G. S. Parnell, and L. J. Lemkuhl. An air mission planning algorithm using decision analysis and mixed integer programming. *Operations Research*, 45(5):662–676, Sep-Oct 1997.
- [6] John Holland. *Adaptation In Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.
- [7] Cezary Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13:189–228, 1993.
- [8] V. C-W. Li, G. L. Curry, and E. A. Boyd. Strike force allocation with defender suppression. Technical report, Industrial Engineering Department, Texas A&M University, 1997.
- [9] Sushil J. Louis. Learning from experience: Case injected genetic algorithm design of combinational logic circuits. In *Proceedings of the Fifth International Conference on Adaptive Computing in Design and Manufacturing*, page to appear. Springer-Verlag, 2002.
- [10] Sushil J. Louis and J. Johnson. Solving similar problems using genetic algorithms and case-based memory. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 283–290. Morgan Kaufman, San Mateo, CA, 1997.
- [11] Sushil J. Louis, J. McDonnell, and N. Gizzi. Learning to improve genetic algorithm performance on job shop scheduling. In *Proceedings of the Genetic and Evolutionary Computation Conference*, page to appear. Morgan Kaufman, San Mateo, CA, 2002.
- [12] Sushil J. Louis, Gary McGraw, and Richard Wyckoff. Case-based reasoning assisted explanation of genetic algorithm results. *Journal of Experimental and Theoretical Artificial Intelligence*, 5:21–37, 1993.
- [13] C. Ramsey and J. Grefenstette. Case-based initialization of genetic algorithms. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 84–91, San Mateo, California, 1993. Morgan Kaufman.
- [14] C. K. Riesbeck and R. C. Schank. *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Cambridge, MA, 1989.
- [15] D Smith. Bin packing with adaptive search. In *Proceedings of an International Conference on Genetic Algorithms*, pages 202–206. Morgan Kaufman, 1985.
- [16] K. A. Yost. A survey and description of usaf conventional munitions allocation models. Technical report, Office of Aerospace Studies, Kirtland AFB, Feb 1995.