OPTIMIZATION ALGORITHMS FOR RESOURCE ALLOCATION PROBLEM OF AIR TASKING ORDER PREPARATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

AHMET ENGİN BAYRAK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

AUGUST 2010

Approval of the thesis:

OPTIMIZATION ALGORITHMS FOR RESOURCE ALLOCATION PROBLEM OF AIR TASKING ORDER PREPARATION

submitted by **AHMET ENGİN BAYRAK** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Prof. Dr. Faruk Polat _____
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Assoc. Prof. Dr. Halit Oğuztüzün
Computer Engineering Dept., METU _____

Prof. Dr. Faruk Polat
Computer Engineering Dept., METU _____

Assoc. Prof. Dr. Veysi İşler
Computer Engineering Dept., METU _____

Assoc. Prof. Dr. Ahmet Coşar
Computer Engineering Dept., METU _____

Dr. Çağatay Ündeğer
General Manager, SimBT Inc. _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: AHMET ENGİN BAYRAK

Signature :

ABSTRACT

OPTIMIZATION ALGORITHMS FOR RESOURCE ALLOCATION PROBLEM OF AIR TASKING ORDER PREPARATION

Bayrak, Ahmet Engin
M.Sc., Department of Computer Engineering
Supervisor : Prof. Dr. Faruk Polat

August 2010, 55 pages

In recent years, evolving technology has provided a wide range of resources for Military forces. However, that wideness also caused resource management difficulties in combat missions. Air Tasking Order (ATO) is prepared for various missions of air combats in order to reach objectives by an optimized resource management. Considering combinatorial military aspects with dynamic objectives and various constraints; computer support became inevitable for optimizing the resource management in air force operations. In this thesis, we study different optimization approaches for resource allocation problem of ATO preparation and analyze their performance. We proposed a genetic algorithm formulation with customized encoding, crossover and fitness calculation mechanisms by using the domain knowledge. A linear programming formulation of the problem is developed by integer decision variables and it is used for effectivity and efficiency analysis of genetic algorithm formulations.

Keywords: Combinatorial Optimization, Air Tasking Order, Genetic Algorithms, Resource Allocation

ÖZ

HAVA HAREKAT PLANI KAYNAK PAYLAŞTIRMA PROBLEMİ İCİN
OPTİMİZASYON ALGORİTMALARI

Bayrak, Ahmet Engin
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Faruk Polat

Ağustos 2010, 55 sayfa

Son yıllarda, gelişen teknoloji askeri kuvvetlere geniş bir kaynak çeşitliliği sağlamaktadır. Bununla birlikte, bu çeşitlilik savaş görevleri açısından kaynak yönetim zorluklarına da sebep olmaktadır. Hava Harekat Görevleri, hava harekatlarının farklı görev türleri için kaynakları en iyi şekilde kullanarak hedeflere ulaşabilmesi için hazırlanmaktadır. Kombinasyonel askeri hedefleri, değişken savaş durumlarını ve kısıtlamalar bir arada düşünüldüğünde; hava operasyonlarında kaynak yönetimi optimizasyonu için bilgisayar desteği kaçınılmaz olmaktadır. Bu tezde, hava harekat planı kaynak paylaştırma problemi için farklı optimizasyon yaklaşımları üzerinde çalıştık ve başarılarını analiz ettik. Alan bilgileri kullanarak özelleştirdiğimiz çaprazlama, değerlendirme ve kodlama yöntemleri ile genetik algoritma formulasyonları önerdik. Tamsayı karar değişkenleri ile geliştirdiğimiz doğrusal programlama formulasyonu ile genetik algoritma formulasyonları için etkililik ve etkinlik analizleri yaptık.

Anahtar Kelimeler: Kombinasyonel Optimizasyon, Hava Harekat Planı, Genetik Algoritmalar, Kaynak Paylaştırma

To my family

ACKNOWLEDGMENTS

I would like to present my deepest thanks to my thesis supervisor Prof. Dr. Faruk Polat for his valuable guidance, motivation and support throughout this thesis study.

My special thanks go to Ayşe Hilal Yergök for her help and support to complete this work and to all my friends who gave me support whenever I needed.

I am very grateful to my family and partners in Minder and MobilG companies for all their patience and tolerance.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

In recent years, evolving technology has provided a wide range of resources for Military forces. However, that wideness also caused resource management and planning problems in combat missions. Resource management problem for a combat mission is composed of various sub-problems encountered throughout the combat missions' planning. One of the main problems of combat planning is resource allocation problem, which defines the operations against enemy targets and aims to optimize the resource allocation efficiency and effectivity. Researches on this problem emerged with scientific approach usage in the organization management. While dealing strategic and tactical problems of military operations in World War II, need for academic study and scientific support raised and Operations Research (OR) activity officially began with created military OR teams [1]. Resource allocation problem is studied within OR domain, which is a common interest of computer scientists and industrial engineers.

Air Tasking Order (ATO) is a plan, prepared for various missions of air combats in order to reach objectives by an optimized resource allocation. Considering both dynamic conditions of a warfare and command and control facilities, an optimal resource allocation should be provided with a fast decision. Long time required for ATO production makes scientific support inevitable for the air force military operations. Computer assistance has been used as a decision support system for 30 years by U.S. Army [2].

Resource allocation optimization problem of military operations has combinatorial aspects with dynamic objectives and various constraints like resource limitations, cost

and transportation [3]. Therefore, resource allocation formulation of ATO preparation can be studied in **Combinatorial Optimization** problem set, which deals with NP - Hard optimization problems [4]. Previous resource allocation studies, mostly formulate the problem as an integer linear programming problem [5, 6]. These studies investigate solutions through varying methods like greedy, branch-and-bound and knapsack approaches. Some researchers concentrated on genetic algorithm formulation of the problem [7].

Military forces cannot use the methods with exponential execution time complexities, because time ambiguity is not acceptable for time critical ATO preparation. On the other hand, optimizing the gain value for an air combat is crucial for combat environment. Requirement for an algorithm that meets these necessities constitute the motivation of this study.

In this study, we study genetic algorithm formulations for resource allocation problem in ATO production. Problem specific crossover, fitness calculation and selection approaches are proposed for genetic algorithm solution of resource allocation problem through this thesis. These approaches mainly concentrate on dividing the genetic problem into smaller genetic problem pieces using domain knowledge. Customized genetic algorithm approaches are implemented and examined through experiments.

Linear programming formulation is implemented for the problem with integer decision variables. Standard greedy and exhaustive search algorithms are implemented. Linear programming, greedy and exhaustive search algorithms' solutions are used for evaluating implemented genetic algorithm customizations.

We extend the framework in [8] to be able to use different solution approaches on the same input problem set. Enhanced framework eliminates the different problem formulations and execution environment. Moreover, that framework has important capabilities from input generation to solution analysis. Varying resource, target and operations can be created based on provided air force capability configuration by the help of random input generator. Generated combat cases can be rendered and persisted for future analysis and solution executions. Adapters for different solution approach formulations allow the usage of same combat case input for solution generation. In case of a need for new solution formulation, a small adapter can be developed

for an easy integration as a result of plug-in strategy used by the framework.

Feasibility and success ratios of each implementation are analyzed and their differences are compared with each other in order to assess their efficiency and effectivity. Input sets with different characteristics are used for those comparisons.

The remainder of the thesis is organized as follows:

Chapter 2 - Background and Related Work gives an overview of Air Tasking Order preparation and resource allocation problem in ATO preparation. Then related studies in literature are described and their approaches are identified in details. Pros and cons of these approaches are discussed. Main algorithms and formulations used in resource allocation studies are described briefly.

Chapter 3 - Our Work presents our research details. The overview of the framework is presented, the input representation scheme is explained, the algorithm implementations and customizations are described in detail.

Chapter 4 - Evaluation of the Results provides a summary of solutions created by implemented algorithms for various input sets and presents experimental results.

Chapter 5 - Conclusion and Future includes the concluding remarks and states the future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter aims to present an overview of Air Tasking Order preparation and resource allocation problem in ATO preparation. It also includes the current studies in literature and their approaches in detail. Main algorithms and formulations used for resource allocation problem are described briefly.

2.1 Resource Allocation Problem in Air Tasking Order Preparation

In warfare, military forces should be flexible to accommodate to dynamic environment conditions for being successful. This flexibility requires the ability of planning and managing the combat dynamically with effective and fast decisions.

> *An **air tasking order (ATO)** involves both a process of assigning military aircraft, long-range missiles, and relevant air defense and information operations resources to missions contributing to an overall air campaign plan, as well as the actual sets of orders to be followed by aircraft and units of aircraft.* [9]

Air Tasking Order (ATO) is used to define management and planning issues of an air attack. Steps followed during ATO production are shown in Figure 2.1.

Diverse resource types of air forces are used in resource planning of ATO. Our study also involves varying attack resources distributed on bases for problem formulation. This variance is achieved by the computer support during input generation. Resources in our subject of interest are listed below:

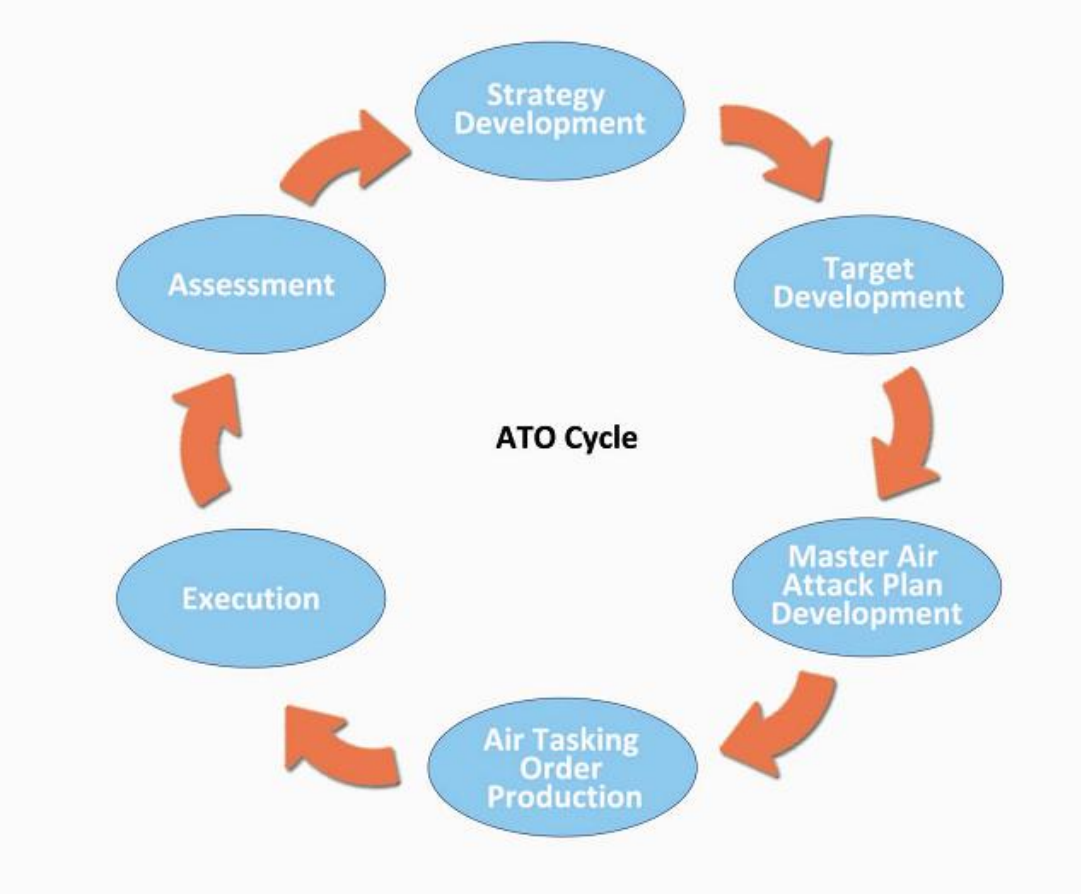

Figure 2.1: Air Tasking Order Production Life Cycle

- **Base:** Military area where all type of resources like munitions, aircrafts, tanks, choppers and staff are located is called base in the literature. Each force (air, naval and ground) has its specific bases and some joint bases.
- **Fleet:** Combatant unit is called fleet in air force bases. Aircrafts and their pilots uniform a fleet unit which is used for target assignment as a group.
- **Aircraft:** Military forces have different types of vehicles used for air combats. They differentiate according to ranges, load capacities and gun capacities.
- **Pilot:** Air combats can be one of two types: air-to-air and air-to-ground combats. Each combat type requires specifically trained pilots who are main staff for a combatant fleet. Pilot capability is one of the key points of combat success as it directly affects the attack potential.
- **Munitions:** Munitions like rockets and missiles are carried by aircrafts in an air combat. As the mass of munitions define their destruction potential, load capacity of an aircraft is directly related with the attack performance.

Each air combat has a procedure from collecting the intelligence information to target destruction. Firstly, enemy resources are examined by related units of air force to collect target information for miscellaneous goals of the military and government. Secondly attack assets; which will be used for enemy target destruction, are prepared by summing up aircrafts, munitions and resources. Each asset resembles a military operation for an air force fleet. In real world's military environment, generally possessed resources are not enough for completing these operations. So, defined operations are analyzed and evaluated according to their relationships with each other. Resource allocation in ATO preparation is finalized with selecting the operations to optimize the combat performance.

ATO resource allocation planning is also performed during a combat according to the realized operations' performance. Considering the both mathematical complexity of resource allocation and time shortness in warfare condition, need for computational decision support system can be seen easily. Computational decision support for resource allocation optimization is one of the active subjects in operations research and computer science domains, which is widely studied [10, 11].

In this thesis work, we elaborated mostly on the allocation problem of scarce resources to varying operations defined according to enemy targets. This problem is also named as ”Weapon Target Assignment” or ”Military Asset Allocation”. In the literature, this resource allocation problem is solved by grouping military attack resources into asset packages and assigning these packages to enemy targets by optimizing the gain from the attack resource potential [8]. Structures of air force targets like airports, buildings, etc. also supports the attack resource grouping strategy as single resource potential are not sufficient for destroying a single target. So, we also analyzed the input problem set in similar way and created operations against enemy targets by grouping resources in the air force.

2.2 Required Features of Resource Allocation Solutions

Proposed solutions for resource allocation problem of ATO preparation should have some features as the real world military environment forces some constraints during an air combat. Requested features are following:

- **Integer solutions:** In real world military environment, we cannot assign half aircraft to a target, so every aircraft should be grouped and assigned as a whole. This constraint affects the usage of algorithms with real number solutions as they cannot be used singly but can be a part of a hybrid solution. In addition, operation necessities should be fully meet as partly completed operations do not have any value for an air combat.
- **Efficiency:** Solution algorithms are expected to run in acceptable time and space. An algorithm with high resource complexity causes inefficient solutions for the bigger sized inputs.
- **Effectivity:** Solution performance of algorithms is very crucial for the problem as they are used in mission critical plans. Gain loss according to best solution is evaluated for an algorithm together with their time acquisitions.
- **Problem formulation:** Each algorithm should have a solution space that maximizes the coverage of all available resource allocation possibilities. Solutions are expected to meet real world military constraints.

2.3 Related Work

Rosenberger et al. [5] formulated the problem as a linear integer programming problem. They investigated two different solution methods for the optimization of military resource allocation problem:

- Greedy Approach: This approach is based on sequential application of auction algorithm. In each step a resource is allocated to a target from the graph. Greediness improves the time performance in this solution.
- A branch-and-bound framework that enumerates feasible tours of assets / resources - a process that can become computationally intensive with increasing number of sources and targets but will find an optimal solution. An optimal solution is tried to be found by enumerating their feasibilities. Branching in this solution methodology works for small sized inputs. If a greedy selection is performed in between branching steps, several suboptimal solutions would be pruned.

Collaborative planning, multiple target assignments for a platform is additional works done in this topic by the researchers. Assigning multiple sources to targets in branch and bound approach seems to make algorithm stronger against greedy approach [5].

Toet and Waard [12] studied the algorithms used for combinatorial optimization problems in military domain. This research is a part of The *EUCLID* (European Cooperation for the Long term In Defence) *CALMA* (Combinatorial Algorithms for Military Applications) *RTP* (Research and Technology Project). Their study explains the Weapon Target assignment problem and different approaches used for solution optimization. Optimization performances of those approaches are analyzed based on:

- Space and time required,
- Ease of implementation and flexibility,
- Quality, computing time relation,
- Stability.

CALMA algorithms are analyzed and their solutions are evaluated according to surviving value maximization and threat minimization approaches [12]. These algorithms aim to optimize resource allocation for a defence force against an air attack. So their objective is maximizing the asset survival and minimizing the attacking enemy threats for a better defence performance.

Ahuja et al. [13] studied Weapon Target Allocation problem as optimization of threat minimization by assigning defense resources against enemy attack threats. Their study is a combination of most commonly used optimization algorithms for WTA problem. A branch and bound algorithm is applied to the problem using the branching lower bounds which are calculated using:

- Linear programming,
- Mixed integer programming (MIP),
- Minimum cost network flow,
- Greedy approach based on combinatorial arguments.

Ahuja et al. proposed a very large-scale neighborhood (VLSN) search algorithm and a construction heuristic that uses the minimum cost flow. VLSN algorithm solves the problem like a partition problem and gives highly impressive results. Problem is divided into partitions and neighborhood search is performed on these partitions. In addition, the MIP lower bounding scheme gives the tightest lower bounds but also takes the maximum computational time [13].

Griggs et al. [14] studied on resource allocation problem of air mission planning and described their research called Joint Stochastic Warfare Analysis Research (JSTOCHWAR). They propose a mixed integer programming with asset grouping strategy which also uses Suppression of Enemy Air Defenses (SEAD) aircrafts against enemy targets.

Aberdeen et al. [10] formulated the planning problem of military operations as a problem of Markov Decision Process (MDP) by considering timing and relationships of military operations. Real-time heuristics that is generated automatically is used in evaluation. MDP formulation is as following:

- **State Space:** States defined by operations, resources, time, etc. about the military operation.
- **Actions:** Military operations that are enabled for the given state.
- **Successor State:** A new state after the performed operation.
- **Costs:** Resources used, targets destroyed, etc. for the state transition.

Aberdeen et al. proposed a *Labeled Real Time Dynamic Programming (LRTDP)* which can be defined as a combination of greedy approach with labeling of converged states [10].

Lee et al. [7, 15] studied genetic algorithm solutions of weapon target assignment problem. In [7], Lee et al. proposed to include domain specific knowledge for the crossover operator and local search mechanism in the genetic algorithm implementation. According to this approach following operations are performed:

- Local search with domain knowledge is performed before and after the mutation of each recombination step of genetic algorithm. They investigated the decrease in time requirement after elimination of weak genomes of the population.
- Fitness value is optimized by the help of Simulated Annealing methodology.

This research [7] mostly emphasizes use of domain specific knowledge in local search and cross-over phases. It is shown that, modification or usage in hybrid solutions can over-perform relative to basic algorithm [7].

In [15], eugenics aided genetic algorithm approach is studied. Genetic algorithm and simulated annealing methods were realized individually and together. Greedy eugenics contribution based on domain knowledge performed better for the problem [15].

McDonnell et al. [16, 17] studied different approaches for genetic algorithm solution of Strike Force Asset Allocation Problem. In [16], a genetic algorithm solution is proposed for resource allocation in strike forces domain. Each genome represents an assignment of a platform to a target in representation. They used CHC selection

methodology described in [18]. In CHC, population is doubled after mutation and crossover operations then best individuals are selected for decreasing the population size to normal.

In [17], McDonnell et al. proposed case based reasoning injection to genetic algorithm for strike force allocation problem in dynamic warfare scenarios. In this proposal following operations are performed in given order:

1. A case database for each recombination in population should be created from warfare scenarios.

2. Some genomes similar to current best solution genome should be added to the population in order to replace previous weak genomes. Those injected new genomes are selected from a case database by the help of a Case Based reasoning component. Matching the relevant case in the database is performed using its hamming distance to a best solution genome in the population.

This novel approach performs better than standard implementation [17].

2.4 Resource Allocation Methodologies

Main algorithms and formulations used for resource allocation problem are described briefly in this part.

2.4.1 Exhaustive Search

Exhaustive search is the methodology that finds a solution for a problem by searching best solution with full coverage on the solution space. Although it has very high time complexity, best solution is guaranteed for this algorithm. In real world military environment, its time consumption is unacceptable so we used this methodology for finding the optimum solution. In resource allocation problem, this algorithm's formulation has $O(2^N)$ complexity. N is equal to T X F where T is the target count, F is the fleet count. However, one assignment for each target constraint decreases this

complexity to O(H^F). Some bounding strategies can be used for decreasing the time consumption of exhaustive search. These strategies are:

- **Branch and Bound:** After creating a single full solution, remaining candidates can be evaluated according to forecasting with their current result quality. A candidate solution is not processed anymore if its forecasted potential is lower than current best solution.

- **Greedy Bound:** Unaffordable operations are removed from solution space and every new candidate is processed greedily with better result quality expectation.

- **Knapsack Bound:** It is used for evaluating remaining candidates. In a Knapsack Problem, there are entities with given size and values and it is expected to select some (one or more) of subsets from these entities for maximizing the sum of entity values without exceeding the total size capacity [19].

2.4.2 Greedy Algorithm

Greedy algorithm is a customization of exhaustive search based on greedy selection of best solution. This methodology does not change selected best result and constructs a single solution at the end of execution. Time complexity of greedy algorithm is O(NlogN). Greedy algorithm can also use different evaluation bounds for resource allocation problem like ones given for exhaustive search.

2.4.3 Integer Linear Programming

Linear programming is used for optimization problems and it is applied on specific problems with a particular formulation described in [20]. Best solutions like maximum gain or minimum cost is found through a mathematical model by the help of domain constraints encoded as linear equations.

Linear programming methodology is widely used in operations research. Elements of linear programming are given below:

- **Linear Objective Function:** Value to be optimized is called objective function. This function should be represented as a linear equation, such as:
  - Maximize: $c_1 x_1 + c_2 x_2$
- **Constraints:** Linear inequalities of the problem domain that bounds the solution space are called constraints. An optimum solution that meets these constraints' requirements is searched by objective function. Each constraint should be represented as a linear equation, such as:
  - $a_{1,1} x_1 + a_{1,2} x_2 \leq b_1$
  - $a_{2,1} x_1 + a_{2,2} x_2 \leq b_2$
  - $a_{3,1} x_1 + a_{3,2} x_2 \leq b_3$
- **Decision Variables:** Both objective function and constraints are based on decision variables x_i whose optimal values are searched with simplex methods in linear programming.

Simplex Method: Basic algorithm generally used for linear programming is the simplex method [21, 22]. It was proven to solve linear formulated problems of acceptable size in a reasonable time.

> *The simplex method works by finding a feasible solution, and then moving from that point to any vertex of the feasible set that improves the cost function. Eventually a corner is reached from which any movement does not improve the cost function. This is the optimal solution.* [21]

The problem is usually formulated in *matrix form*, and represented as:

- Maximize: $c^T x$
- Subject to: $Ax \leq b, x \geq 0$

where **x** represents the vector of variables (to be determined), **c and b** are vectors of (known) coefficients and **A** is a (known) matrix of coefficients [23]. In this formulation, a vector **x** is a feasible solution of the linear programming problem if it satisfies the given constraints. Problems defined in this formulation have three different types [21]:

- **Infeasible:** None of the vectors in solution space can satisfy the given constraints.
- **Unbounded:** Given constraints are not enough for bounding objective function parameters in the solution space. So, a better solution with an improved objective function value can exist.
- **Optimal:** Problem formulated in linear programming has an optimum value for the objective function and there exists vector(s) that can create such optimum value with satisfying the given constraints.

Integer linear programming is a customized version of linear programming where all decision variables of objective function are *integers*. As resource allocation problem requires integer results, only integer linear programming can be used for finding the solution.

2.4.4 Genetic Algorithm

Genetic algorithm is one of the powerful and widely used evolutionary computation methodologies for optimization and search problems [24, 25]. Genetic algorithm does not create a problem specific result. However, it proposes a general formulation for every problem by searching solution space with defined relevance calculation. Used randomness and statistics cause different solution for different executions of the algorithm so more than one trial can be made for gathering best result. Main terms used in this formulation are as followings:

- **Genome:** Each individual solution is defined as genome so each solution should be able to be represented as genome in a genetic algorithm. Genome encoding has a critical importance for algorithm performance. Encoding can be binary or numerical in any dimension (1D, 2D, etc.).
- **Population:** Genomes in solution space constitutes a population. Genetic algorithm aims to maximize the individual solution performance in a defined population by the information from each other and from the outside. This performance improvement is realized by genetic operations step by step.

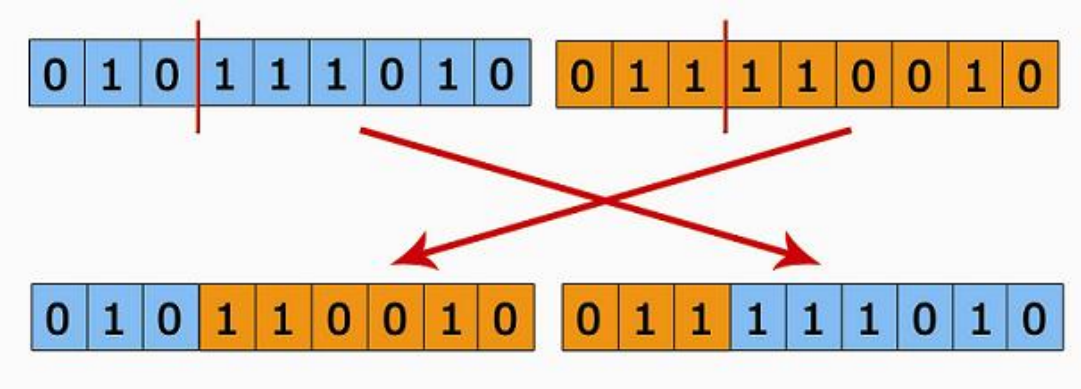

Figure 2.2: Single Point Crossover

- **Crossover:** In each step, genetic information of population genomes are shared by genetic crossover operations over selected random individuals. Offspring genomes are created from existing genomes by that way. Procedure for single point crossover operation is shown in Figure 2.2:

- **Mutation:** Closed population has the risk of being caught by local maximum value after some individual genome's quality improvements because total information shared has the limitation for a closed population. In order to escape from these local maximums, some outside information can be added to population. Mutation is one of the outer information addition strategies, where some parts of randomly selected genomes mutated.

- **Fitness Value:** Relevance calculation used for each genome's evaluation is called fitness function. Return value of this function is called fitness value which is used as a primary property in the selection step.

A random search is performed in this algorithm where crossover and mutation operators help the searcher in heading towards the population's subset which optimizes the objective value. This help should be in such a way that satisfies the equilibrium between exploration and exploitation of the population space. Therefore, such operators and population encoding should be examined carefully when formulating the genetic algorithm solution for the specific problem [24]. Steps of a typical genetic algorithm is given in Figure 2.3 where **P(t)** represents the solution population for generation **t** and **C(t)** represents the created offsprings for the t^{th} generation:

```
Procedure: Genetic Algorithms
begin
 t ← 0;
 initialize P(t);
 evaluate P(t);
 while (not termination condition) do
 begin
  recombine P(t) to yield C(t);
  evaluate C(t);
  select P(t + 1) from P(t) and C(t);
  t ← t + 1;
 end
end
```

Figure 2.3: Genetic Algorithm Procedure

Main problems that should be solved for genetic algorithm adaptation of a problem are encoding and selection operators.

2.4.4.1 Encoding

A solution instance is represented as genomes in genetic algorithms and this representation is realized by the help of selected encoding methodology. Problem requirements and solutions spaces vary in operations research domain. Some of the real world problems can be defined with binary encoding but some problems' complexity does not allow binary encoding usage. Encoding classifications *according to used symbols* are:

- **Binary Encoding:** It is generally used because of simple encoding which provides easiness in selection of operators [26].
- **Real Number Encoding:** Used for optimization problems as real number usage is identical to the problem's real world representation [27].
- **Literal Permutation Encoding:** Used for problems that search for a best combination or permutation of the parameters.
- **Data Structure Encoding:** Used for complex problems that cannot be represented with other encodings. A data structure related with problem domain is

used.

Encodings are also classified *according to genome dimensions* as:

- **One Dimensional Encoding:** Used for problems that does not lose information. It has a low selection operator complexity.
- **Multi Dimensional Encoding:** Used for the complex real world problems that loses information in one dimensional encoding [28]. A matrix is used as genome so selection operator complexity increases.

Adaptation can be made with a suitable encoding methodology defined above. Any encoding methodology chosen for a problem formulation should have following properties to be effective [24]:

1. **Complete:** Any solution should be able to be represented in the solution population with encoding.
2. **Legal:** Encoding permutation of a solution should yield another solution in population.
3. **One-to-One Mapping:** Encoded genomes should not be redundant with real solutions. One representation exists for one solution instance.
4. **Heritable (Lamarckian):** Genomes should not have context dependent properties. Genome properties should pass from parent to offspring [29].
5. **Causal:** This property defines the effect of genome mutation on the real world solution representation. It is preferred to have small changes in real world representation when a genome representation has a small change [30].

2.4.4.2 Selection Operators

In genetic algorithm, solution space which is called population is searched for a genome with best objective function values. There are two approaches in the searching methodology:

- **Random Search:** Population is explored entirely and local maxima can be escaped.

- **Local Search:** Solution space is searched with optimum genome exploitation and local maxima cannot be escaped.

For a good solution, search in population should satisfy the equilibrium between exploring whole solution space with random search and accumulated information exploitation with local search [24]. Crossover operator defines the offspring generation in genetic operation. Mutation operator determines the random changes over population. Crossover and mutation operators play a critical role in satisfying that search equilibrium. In the literature following methodologies used for adding exploitation power to the random search:

- **Building Blocks:** This genetic approach is developed by Holland [26] and improved by Goldberg [31]. It emphasizes on the attribute heritage from parent genomes to offspring genomes by recombining genome parts called genes with crossover operator. Quantity of feature heritage determines the offspring attribute quality calculated as fitness value. These fitness values are generally calculated by fitness functions which are using both feature values alone and together as patterns. Building blocks approach is applicable for the genome representations in which effect of a gene singly on fitness value is more than effect of gene patterns on fitness value. So, genes with dependency to other genes in fitness value calculation are not suitable for this recombination of features approach.

- **Convergence Controlled Variation:** This approach is developed by Eshelman and his friends [32]. Genetic search is directed according to solution convergence of the population. Exploitation is realized by extending a population in the way of a preferred convergence distribution such that solution space variation becomes more focused on convergence to objective.

Exploitation strategy is determined according to problem formulation and its genome encoding. One other step of genetic algorithm is the selection of genomes. Fitness

value calculation becomes critical in this step as the future generations of the population is dependent on the selected genome. Generally fitness values are defined with considering both the problem objective and the genetic formulation used. Selection mechanism is another property that uses and affects the fitness function. Some of the commonly used selection mechanisms in the literature are followings:

- **Roulette wheel selection:** It is a random selection for gene recombination. In this mechanism, a genome is randomly selected for crossover from a pool filled according to a probability based on genome's fitness value. Percentage of a genome's fitness value defines the possibility of that genome in the pool.
- **Tournament selection:** Tournaments are prepared between genome instances and the winner is selected.
- **Steady State selection:** In this mechanism, survival of the current populations' some parts is achieved.
- **Elitism based selection:** Best solution of the current population survives by copying to new population.

Effects of the selection mechanism choice can be observed from the minimum generation count required for that implementation.

CHAPTER 3

OUR WORK

This section presents details of our research about resource allocation problem in ATO preparation. First, problem formulation and scenario assumptions are defined. Then, input generation and input sets are explained. Finally, architecture of used framework, algorithm implementations and customizations are described in detail.

3.1 Problem and Solution Formulation

In this study resource allocation problem is solved as selecting a subset from the defined operation list in order to maximize the gain from targets with the possessed air attack resource potential. Gain is calculated as sum of all destroyed targets' gaining.

An air combat's goal is to destroy enemy targets whose classification, location and gain value information are defined in intelligence information acquired. An operation with calculated risk and gain value is defined against a single target in our study. Selecting an operation list subset with more than one operation dealing with the same threat does not improve the total gain of the solution as one target can be destroyed once in real world. The input format is shown in Figure 3.1.

In the initial operation list, each operation is generated according to resource requirements of a target. This requirement is defined in terms of attack resource capabilities and operations represent these requirements in terms of aircraft counts from a fleet. There can be more than one operation in a selected operation list subject with the same resource fleet. However, number of aircrafts in all operations in a selected subset cannot be more than available aircrafts in the related fleet. Considering the combinatorial

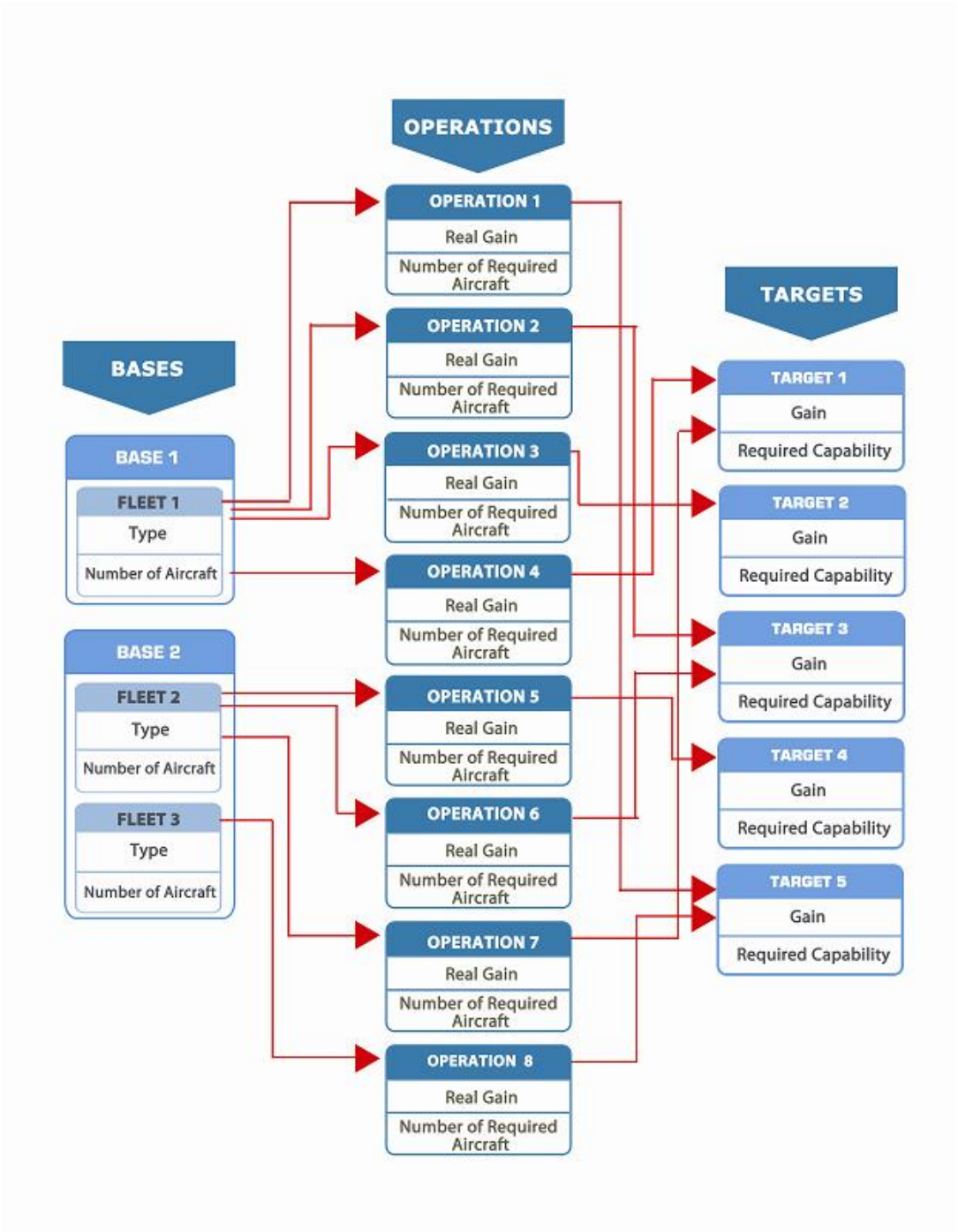

Figure 3.1: Problem Input Format

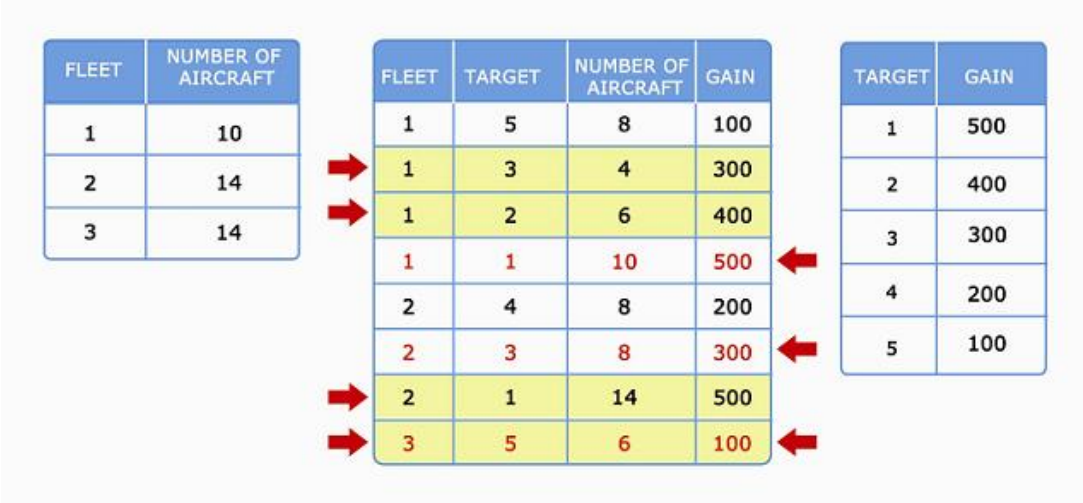

Figure 3.2: Problem Instance with Solutions

constraints and the allocation specifications, some real world parameters are not used for decreasing complexity by scenario assumptions described in Section 3.3.

In this thesis, solutions found by different algorithms and approaches are expected to optimize the gain of the selected subset from the given operation list. Each operation is performed by aircrafts of single fleet and against single target. Solutions with requiring more than fleet's aircraft capacity is not considered as well as solutions against the same targets.

An instance of the resource allocation problem with solutions are shown in Figure 3.2. If that problem is solved greedily, solution will be a subset consisting of operations with an arrow on the left side of the table. However, an optimized solution is available if a subset is created from operations which have an arrow on right side of table.

3.2 NP-Completeness of Problem

After defining the problem formulation, problem properties are analyzed in order to be able to select appropriate solution methodologies accordingly. Main property analyzed for algorithm implementation is problem's complexity with the help of computational complexity theory [33].

In complexity theorem, a problem is examined with varying sizes to decide on the problem behaviour. This behavior is evaluated by applying the algorithms to problem instances and analyzing the algorithm execution time. An algorithm is called **polynomial time algorithm** if maximum value of this execution time can be defined within terms of a polynomial function. Using algorithm's execution time formulation, each problem is classified as an element of one of the following sets:

- **P Class:** Elements of this set has a polynomial time algorithm implementation that can find a solution in polynomial time, working on a deterministic machine.
- **NP Class:** Elements of this set has a polynomial time algorithm implementation that can only find a solution in polynomial time with a non-deterministic machine. A machine with unlimited parallelism is called non-deterministic. Well known combinatorial optimization problem *travelling salesman* is an NP problem as all solution possibilities of the problem can only be examined with a non-deterministic machine in polynomial time [34].

Relation between **P** and **NP** Set can be defined as $P \subseteq NP$. However, it is not proved whether P is equal to NP or P is a subset of NP. Hardest problems in NP set form a NP - complete set. This problem set is defined for some NP problems using the convertibility property such that a NP problem can be transformed to any other with a conversion algorithm working in polynomial time. This property shows us that any NP - complete problem is at least as hard as all other NP problems. Therefore, if there exists an algorithm with polynomial time *(P)* for a NP - complete problem, there is a similar algorithm for all NP problems. We can say that NP - complete problems cannot be solved in polynomial time *(P)* until NP and P sets' equality is proved.

The optimization problem in our research can be proved to be an element of NP - complete problem set according to complexity theorem:

- We define the problem formulation as decision problem for examining whether gain of a solution is greater than or equal to a value **X**:

$$\forall t \in T \left(\sum_{o \in O', d(o)=t} 1 \right) \leq 1, \forall f \in F \left(\sum_{o \in O', s(o)=f} k(o) \right) \leq c(f), \sum_{o \in O'} g(o) \geq \mathbf{X} \quad (3.1)$$

- O is the set of all operations
- O' is the compared problem solution, $O' \subset O$
- F is the set of all fleets
- T is the set of all targets
- s(o) is the fleet source used for the operation o, $s(o) \in F$
- c(f) is the number of existing aircrafts in fleet f, $c(f) \in Z^+$
- d(o) is the destination target for the operation o, $d(o) \in T$
- k(o) is the total number of aircrafts used for the operation o, $k(o) \in Z^+$
- g(o) is the gain of the operation o, $g(o) \in R^+$

- First requirement of NP - completeness, **being a NP problem** is valid for defined problem. Because, calculating gain value of a solution instance and checking the solution validity can be performed in polynomial time.

- Then we find a NP - complete problem that can be reduced to our problem in polynomial time to prove **the convertibility** requirement of NP - completeness defined above. Well known NP - complete problem, *0-1 knapsack decision problem* [35] is examined:

$$\sum_{v \in V'} w(v) \leq \mathbf{W}, \sum_{v \in V'} p(v) \geq \mathbf{X} \tag{3.2}$$

  - X is the objective value
  - V is a finite set of objects
  - W is the total weight that can be carried with the knapsack
  - w(v) is the weight of the object v, $w(v)) \in Z^+$
  - p(v) it the gain of the object v, $p(v) \in Z^+$

- Reduction of 0-1 knapsack problem to our resource allocation problem can be achieved fully by following formulation:

  - Single fleet assumption by F = f and representing all aircrafts in fleet by *c(f) as W* of knapsack
  - Each target is represented as an object, T = V

- Single operation for each target assumption (same identification by d(o) = o) so representing each operation as an object, O = V
- Every operation is performed with the single fleet, $s(o) = f$
- Number of aircrafts used for an operation is represented as weight of an object, $k(x) = w(v)$
- Gain of an operation is represented as gain of an object, $g(o) = p(v)$
- Total gain of solutions are equal and represented with **X**

As our problem verifies the requirements of NP - completeness, we can say that resource allocation problem is an element of NP - Complete problem set.

3.3 Scenario Assumptions and Restrictions

Different algorithms used in this thesis have different problem forms. In our study, some assumptions are made and some restrictions are determined in order to be able to use the same input scenario for all implemented algorithms. These assumptions and restrictions simplify the algorithm formulation and execution. They are listed below:

1. Each pilot in a fleet has the same capability and each fleet has single type aircrafts. Single type aircraft restriction in a fleet can be adapted to real world military environment by creating sub fleets according to pilot capability or aircraft type differences.
2. Fleet capabilities are defined for each operation type according to all aircrafts and pilots in the fleet.
3. One operation should be able to destroy the target. Target, that requires two operations, is not acceptable for our study. However, such requirement in an air combat can be adapted to the framework by propagating targets with virtual target addition.
4. Time and dependency relationships of operations are ignored in resource allocation problem by the framework. In case of a scenario requirement with

such relationships, layered problem architecture can be used after dividing the problem into sub problems.

5. Only resources in a fleet are used for the algorithms. However, other resources in bases can be adapted to our formulation by virtual fleet addition for such resources.

Our assumptions do not change the problem structure. Simplified representation still covers most of the real world air combat scenarios.

3.4 Input Generation and Input Sets

Using the same input for various algorithms is one of the key points of our study as it gives us a chance of better evaluation and comparison. Moreover, our framework lets us future usage of the same input for another execution. The input is randomly generated and maintained by the tool. Properties of the generated input set play a critical role for the algorithm execution according to the following main parameters:

- **Fleet and Target Count:** Each algorithm has its own complexity and these values are the variables used in complexity calculations. Increase in these parameters directly effects efficiency and some algorithms like exhaustive search.
- **Operation List:** Operation list is automatically generated according to requirements of the randomly generated targets and potential capabilities of the fleets. It is important for some problems such as genetic algorithms as genome length is directly proportional with the operation list size.
- **Required and Provided Capacity Rate:** Success possibility of each generated input is represented by this capacity rate. For example, if the required aircraft count is 60 for enemy targets and air force has 30 aircrafts for that combat then probably half of the targets would be destroyed.

Playing with parameters given above, input scenarios with different characteristics are generated. This diversity is needed because of exhaustive search algorithm's complexity and demand for larger inputs. Moreover, linear programming has also high

complexity for the studied problem as integer decision variables required. Other algorithms and their customizations also needed to be evaluated with various inputs. Input sets created for this study depend on size and capacity rate differences;

- Small sized input set applicable for exhaustive search and linear programming,
- Input set with high capacity rate and
- Input set with low capacity rate.

3.5 Framework Architecture

In this thesis, the framework used is a plug-in based algorithm execution tool for resource allocation problem. Scenarios are created by Random Input Generator tool with parameters described before. An air combat scenario passes through following path until the comparison step:

1. **Input Encoding:** Every algorithm added to framework as a plug-in should have an encoding unit that takes the well defined common input representation and encodes in the way algorithm requires.
2. **Algorithm Configuration:** Each algorithm has its own parameters and specific configuration. Framework gives a graphical user interface for such configurations to the user. Customizations are directly used in execution.
3. **Algorithm Execution:** Framework runs added algorithms (plug-ins) on the same input scenario. This execution details are not known by framework so each plug-in determines the execution steps.
4. **Result Collection:** Each algorithm is expected to return its result by a well defined interface to a table for analysis.

Any new algorithm can be added if it implements 4 steps of execution path.

3.6 Algorithm Implementations and Customizations

This part defines the algorithms and customization made for them for this thesis work.

3.6.1 Exhaustive Search

All instances in solution space are evaluated in exhaustive search algorithm. Studied problem is represented as examining all operations for each target and finding the optimum in the solution space. In this thesis, none of the bounding approaches is used for decreasing the execution time for this algorithm, because we implemented exhaustive search to create reference optimum solution values for small sized input scenarios. Following is the general structure of exhaustive search algorithm implementation for resource allocation problem:

1. Input:
    - A set of operations **O**; representing all operations for the problem instance.
    - A set of targets **T**; representing all targets for the problem instance.
    - A set of fleets **F**; representing all fleets for the problem instance.
2. Output:
    - A set of operations **S** representing operations that will be performed from the **O** set for optimization, $S \subset O$
3. ***ExhaustiveSearch (O, T, F)*** function that examines all subsets of O and returns with the maximum gain.

3.6.2 Greedy Search

Solution space is explored greedily in this algorithm. Operations are sorted according to goal in a list and each operation from the list is examined and added to solution subset one by one. Any operation requiring more than existing fleet resource is passed

and no change is tried on previously generated solution subset for such situations (greediness).

Greedy search algorithm formulation of resource allocation problem is a two step process:

- **Sort:** Solution sorting step is executed first according to goal of the algorithm. In our problem operations are sorted according to gain values as our aim is to maximize the gain value. This takes **O(NlogN)** where **N** represents the number of elements in searched solution space of operations.
- **Examine:** Solution examining step is executed last. Each solution instance is analyzed according to resources and targets. Therefore, this step has a complexity of **O(N)** where N represents the number of elements in the solution space.

Dominant complexity forces greedy search algorithm to *O(NlogN)* complexity.

3.6.3 Integer Programming

Integer programming algorithm is implemented based on the well known free ***lp solve*** library for resource allocation problem [1]. We can define ***lp solve*** as a library which performs a set of linear programming routines and provides an Application Programming Interface (API) that can be called from customized codes for custom problems. Simplex method is supplied by this library which is integrated with a developed plug-in of the framework. Formulation of the problem is given below:

- **Encoding:** Problem encoding is achieved by creating a vector from the **binary** decision variables. This vector has a size equal to all operations defined. A sample vector $\vec{x}$ is defined as in the following equation:

$$\vec{x} = x_1 x_2 x_3 ..... x_K \tag{3.3}$$

  - K is the total number of operations in the defined operations set

[1] http://lpsolve.sourceforge.net/5.5/

- x_N is a binary (integer) decision variable with lower bound of *0* and upper bound of *1*
- Each decision variable x_N in the vector is matched with N^{th} operation in the defined operations set
- If value of a decision variable is *0*, related operation is not performed
- If value of a decision variable is *1*, related operation is performed.

- **Objective function:** The resource allocation problem aims to optimize the gain for an air combat by selecting a subset from the defined operations set. So, integer programming formulation's objective function is the maximization of the total gain calculation as given in the following equation:

$$max \sum_{o \in O} c_o x_o \tag{3.4}$$

where O is the set of all operations, c_o is the gain from the operation o and x_o is the value representing the operation o in the decision variables vector $\vec{x}$.

- **Constraints:** Any solution of the resource allocation problem should have some defined features to be valid. Those features can be formulated by the constraints term in the linear programming. Following items define constraints implemented in our study:

  1. **Resource Bounding:** Each solution should guarantee not to use more than existing resource of the fleets. For each fleet, sum of performed operations' requirement for that fleet cannot exceed the resource count. To control the resource requirements, a vector $\vec{rr}$ is created for each fleet:

$$\vec{rr} = rr_1 rr_2 rr_3 ..... rr_K \tag{3.5}$$

     - K is the total number of operations in the defined operations set
     - rr_N is an integer value that represents the resource requirement of the N^{th} operation in the defined operations set.

     In order to add resource bounding constraint for a fleet f_i, created vector $\vec{rr}$ is used in the following constraint equation:

$$\sum_{o \in O} rr_o x_o \leq rc(f_i) \tag{3.6}$$

- O is the set of all operations
- rr_o is the resource requirement representation of the operation o from the i^{th} fleet in $\vec{rr}$
- x_o is the value representing the operation o in the decision variables vector $\vec{x}$ and
- $rc(f_i)$ represents the total resource capacity of the i^{th} fleet

Vectors created for all fleets are used in composing coefficient matrix **A**:

$$A = \begin{bmatrix} \vec{rr^1} \\ \vec{rr^2} \\ . \\ . \\ \vec{rr^K} \end{bmatrix} \tag{3.7}$$

where K is the total number of fleets and $\vec{rr^N}$ is the created resource requirement vector for N^{th} fleet.

Matrix **A** is used in well known $A\vec{x} \leq \vec{b}$ constraint equation of linear programming, where $\vec{b}$ vector is filled with the resource capacity $rc(f_i)$ values.

2. **Single Destruction:** Each solution for resource allocation problem should guarantee not to perform more than one operation for the same target. In order to add this single destruction constraint, a vector of binary values is created for each target.

$$\vec{dt} = dt_1 dt_2 dt_3 .....dt_K \tag{3.8}$$

- K is the total number of operations in the defined operations set
- dt_N is a binary value that represents the relevance of the N^{th} operation with the related target.
- *I*f dt_N is 0, N^{th} operation does not destroy the related target.
- *I*f dt_N is 1, N^{th} operation destroys the related target.

Vectors created for all targets are used in composing coefficient matrix **A**:

$$A = \begin{bmatrix} \vec{dt}^1 \\ \vec{dt}^2 \\ . \\ . \\ \vec{dt}^K \end{bmatrix} \tag{3.9}$$

where K is the total number of fleets and $\vec{dt}^N$ is the created resource requirement vector for N^{th} fleet.

Matrix **A** is used in well known $A\vec{x} \leq \vec{b}$ constraint equation of linear programming, where $\vec{b}$ vector is filled with binary value ***1*** for each target as single destruction is requested at most.

Simplex method of the integrated library uses the constraints given above and aims to find the **x** vector representing the operation realizations which maximizes the total gain of resource allocation.

3.6.4 Genetic Algorithms

In genetic algorithm implementations of optimization problems, the proposed general formulation is adapted by deciding on the representation and calculation terms of the algorithm according to the real world problem. During our study, we implemented different approaches for specific terms of genetic algorithm like encoding, crossover, etc. Resource allocation problems are adapted according to these implementations and solution qualities are analyzed for each approach for terms. Two different strategies are used in this approach:

- **Simple Exploring Strategy:** Genetic algorithm is used without much customization. Main genetic algorithm properties are kept to constitute a reference solution for the genetic algorithm customizations.

- **Divisive Greedy Strategy:** In order to optimize the solution, operations are divided into sub solutions (sub-genomes) according to the required fleet resource

of operations. Search and selection are performed more greedily than simple exploring strategy by increasing the exploitation rate.

Besides the used strategies, resource allocation problem is represented as a genetic algorithm problem by specifying elements of the genetic algorithm given below:

3.6.4.1 Encoding

Binary encoding is used for representing genomes. Each genome is a one dimensional, fixed length bit sequence. Each bit represents the execution of corresponding operation. Length of each genome is equal to the number of operations in the problem. Binary encoding, fixed length and one dimensionality provide easiness and low complexity for the genetic operations. However, each genome contains useless bits for the operations which are not executed.

Each genome can be represented as:

$$G = E_{o1}E_{o2}E_{o3}.....E_{oK} \tag{3.10}$$

where:

- K is the total number of operations in the problem,
- E_{oN} represents the execution status of N^{th} operation,
- Operations with corresponding bit with value 1 are executed and 0 are not executed.

An example genome is g = '01011001000' and g represents a solution of resource allocation of resource allocation problem by defining a set of executed operations which is a subset of all operations. This encoding is complete as all solutions in solution space can be represented by a genome. Bit ordering in each genome is implemented with two different approaches:

1. **Simple Maximum Gain Encoding:** All operations (bits) in solution space are ordered according to the gain value of corresponding operation. Dependency of

each operation to another is discarded and heritage is expected from crossover operation according to *Building Blocks Hypothesis* [26, 31].

2. **Fleet Based Maximum Gain Encoding:** Bits are ordered according to corresponding operation's required fleet identifier. By that way, all operations requiring the same fleet are grouped together through encoding. Bits in the same group are ordered according to gain values. Each genome becomes a composition of sub-genomes, each representing an operation group as:

$$G = SG_{f1}SG_{f2}SG_{f3}.....SG_{fK} \tag{3.11}$$

where:

- **K** is the total number of fleets in the problem,
- SG_{fN} represents the sub-genome created for N^{th} fleet,
- Each sub-genome is filled with binary values that represent execution status of corresponding operation,
- A sub-genome for fleet **X** is formulated as:

$$SG_{f_X} = E_{o_x1}E_{o_x2}E_{o_x3}.....E_{o_xK} \tag{3.12}$$

where E_{o_xN} represents the execution status of N^{th} operation of the sub-genome created for X^{th} fleet.

3.6.4.2 Crossover

Crossover is the key point of the solution space search. Three approaches are implemented for the crossover term of genetic algorithm:

1. **Single Point Crossover:** A random point from genome is selected in this implementation. Offsprings are generated from parent genomes' fraction according to that randomly selected point. Random search is performed with the exploration effect of random crossover point choice in this approach. Attribute heritage is expected from parent genomes' recombining. Figure 3.3 explains single point crossover.

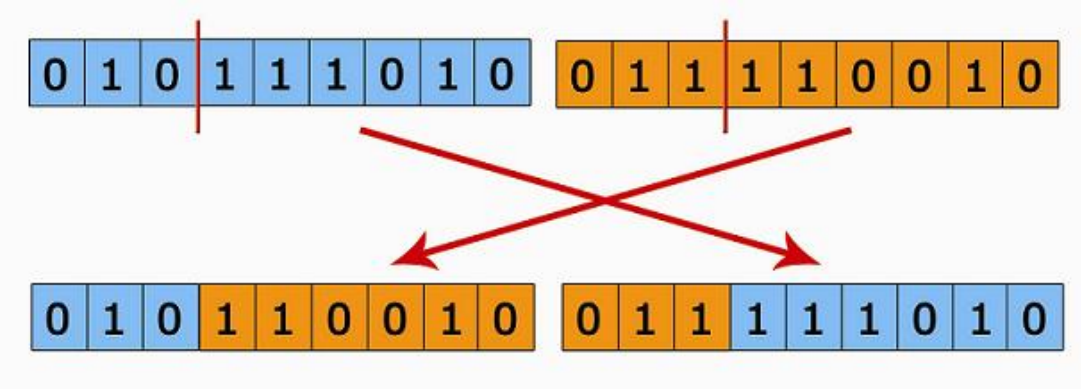

Figure 3.3: Single Point Crossover

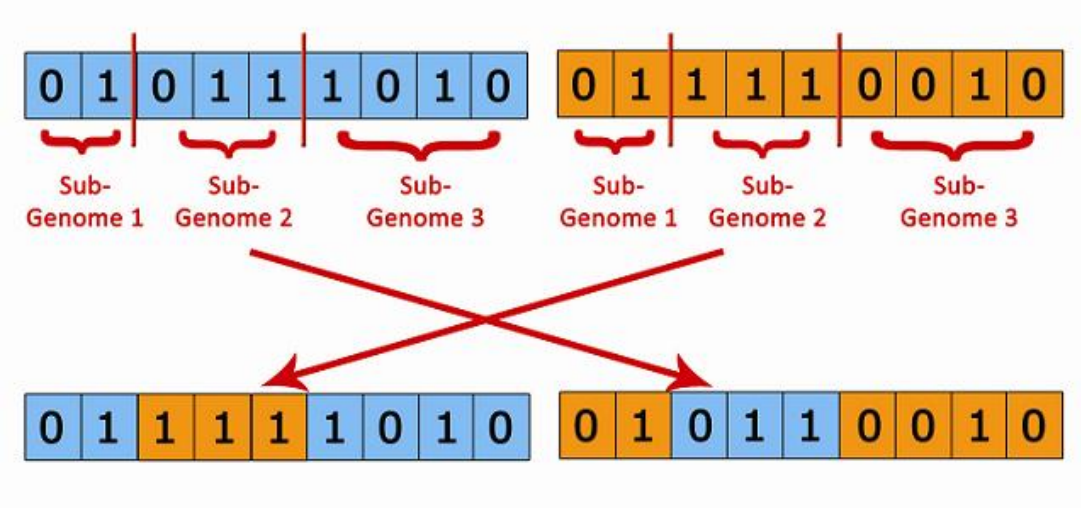

Figure 3.4: Boundary Points Crossover

2. **Boundary Points Crossover:** This crossover is applied to the genomes with fleet based encoding. Points are not randomly selected in this implementation. Selected crossover points are the boundary points between two sub-genomes. Crossover point count is one less than fleet count because count of sub-genomes is equal to the fleet count. Figure 3.4 exemplifies the boundary points used for the crossover. Additional local search behaviour is added to the random search of genetic algorithm by grouping bits. Convergence randomness is decreased by not selecting random crossover points. Attribute heritage in this implementation contains both feature propagation and feature pattern propagation to offspring. In this heritage, sub-genome (bit pattern) with higher gain is propagated

to children to use the *convergence controlled variation hypothesis*. In order to avoid local maxima solutions and achieve equilibrium between random search and local search, randomness is added for the selection point. With fifty percentage probability, sub-genome selection is performed as in the single point crossover implementation.

3. **Layered Hybrid Crossover:** This is an experimental approach with two layered crossover action. This crossover is applied to the genomes with fleet based encoding.

   (a) First step is similar to the single point crossover with randomly selected single point recombining. However, this first step crossover is applied to each sub genome separately. Propagated two offsprings are the combination of created offsprings of sub-genomes. These offsprings are used as a parent in second step.

   (b) Second step is similar to the boundary points crossover. Only difference is that there are four parent genomes: two real parents and two parent from offsprings of the first step. The same crossover action of boundary point crossover approach is applied. However, best two parents are selected from four parents (two real, two virtual) before applying the crossover.

   Layered hybrid crossover methodology aims to use best features of other two crossover methodologies by applying them in different layers. This approach is shown in Figure 3.5.

3.6.4.3 Mutation

Single bit mutation is used in all implementations of the genetic algorithm. Mutation rate is parametric.

3.6.4.4 Fitness Value

Two fitness functions are analyzed with genetic algorithm implementations:

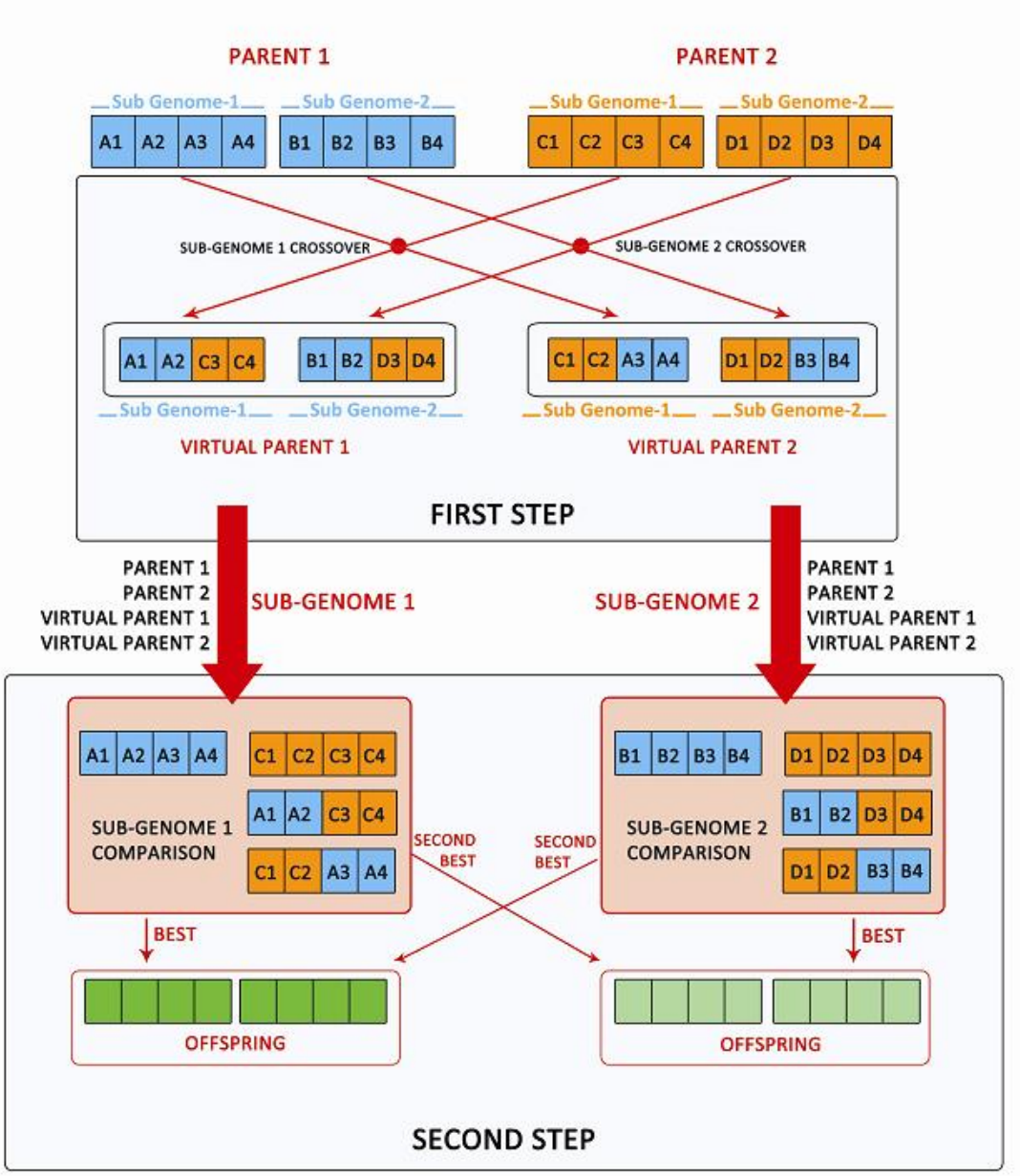

Figure 3.5: Layered Hybrid Crossover

1. The first one is the maximum gain objective function that sums the performed operation gain values. Simple fitness calculation is performed for genome G as in equation 3.13.

$$simpleFitness(G) = \sum_{o \in G} gain(o).isExecuted(o) \tag{3.13}$$

Where gain(o) is the gain of related operation and isExecuted(o) is the execution status of the related operation.

2. The second is a customized version of objective function that looks for the sub-genomes' fitness by the gain percentage for a specific fleet. Main idea in this function is to support genomes with high performing sub-genomes rather than a total good performance. Fitness value of a genome G, which is represented as $G = SG_{f1}SG_{f2}SG_{f3}.....SG_{fK}$, is calculated as in equation 3.14.

$$totalFitness(G) = \sum_{SG \in G} subFit(SG) \tag{3.14}$$

where SG is the sub-genome created for fleet **f** and subFit(SG) is calculated as in equation 3.15.

$$subFit(SG) = \frac{simpleFit(SG)}{allGain(SG)} * \frac{aircraft(SG)}{craftCount(f)} \tag{3.15}$$

Using the utility functions in following equations:

$$simpleFit(SG) = \sum_{o \in SG} gain(o).isExecuted(o) \tag{3.16}$$

$$allGain(SG) = \sum_{o \in SG} gain(o), \tag{3.17}$$

$$aircraft(SG) = \sum_{o \in SG} fleetRequired(o) \tag{3.18}$$

where craftCount(f) is the total number of aircrafts in the fleet **f** and fleetRequired(o) is the number of aircrafts used for operation **o**.

CHAPTER 4

EVALUATION OF THE RESULTS

This chapter provides a summary of solutions created by implemented algorithms and presents experimental results. Evaluations are performed according to objective function values for the input set. Configuration parameters used for implementations are determined according to similar studies in literature and experiments made. These parameters can be listed as:

- **Exhaustive Search:**
  - No bounding
  - Maximum gain based sorting
- **Genetic Algorithms:**
  - Crossover probability = 0,85
  - Generations = 100
  - Mutation probability = 0,1
  - Population count = 1
  - Population size = 400
  - Replacement ratio = 0,4
- **Greedy Algorithm:**
  - Maximum gain based sorting
- **Integer Linear Programming**
  - Integer valued parameters

INPUT SCENERIO		GA_SIMPLE		GA_GROUPED		GREEDY		LINEAR		EXHAUSTIVE	
Operations	Capability	Gain	Time(s)	Gain	Time(s)	Gain	Time(s)	Gain	Time(s)	Gain	Time(s)
8	0,661	3330,83	0,17	3330,83	0,31	3330,83	0,02	3330,83	0,12	3330,83	0,04
20	1,197	5223,06	0,28	5223,06	0,33	4798,12	0,07	5223,06	0,19	5223,06	0,02
36	0,777	11909,00	0,23	11909,00	0,30	11841,00	0,04	11909,00	1,49	11909,00	1,16
54	0,865	11200,00	0,21	11200,00	0,50	10775,00	0,07	11200,00	9,64	11200,00	3,05
64	0,897	17681,00	0,32	17681,00	0,28	16878,00	0,01	18011,00	786,36	18011,00	389,97
75	1,880	12486,80	0,71	12486,80	1,00	12486,80	0,30	NA	NA	12486,80	567,17

Figure 4.1: Results of Input Set 1

- Verbose level = 3
- epsel = 1.0e-11
- epsd = 1.0e-8
- epspivot = 1.0e-5
- epsb = 1.0e-9
- epsint = 1.0e-3

Results are presented separately for each type of input set in the following sections.

4.1 Results of Small Sized Input Set

The time complexity of exhaustive search and integer linear programming avoids us to examine all input scenarios. So some specific scenarios with small sized data with high capacity rate are produced in order to be able to compare algorithms' effectivity. Exhaustive search will be a reference for calculating the gain loss of an implementation. Figure 4.1 shows the generic result values of this input set.

Most important difference is the required time change according to input size. These values depend on the time complexities of the algorithms. Figure 4.2 shows elapsed time values for first three input scenarios of this input set.

Vertical values are the elapsed time of each algorithm for corresponding operation size. Other three inputs are not used in the Figure as the time difference improves so fast to view the complexity variance. Analysis for each implementation is listed below:

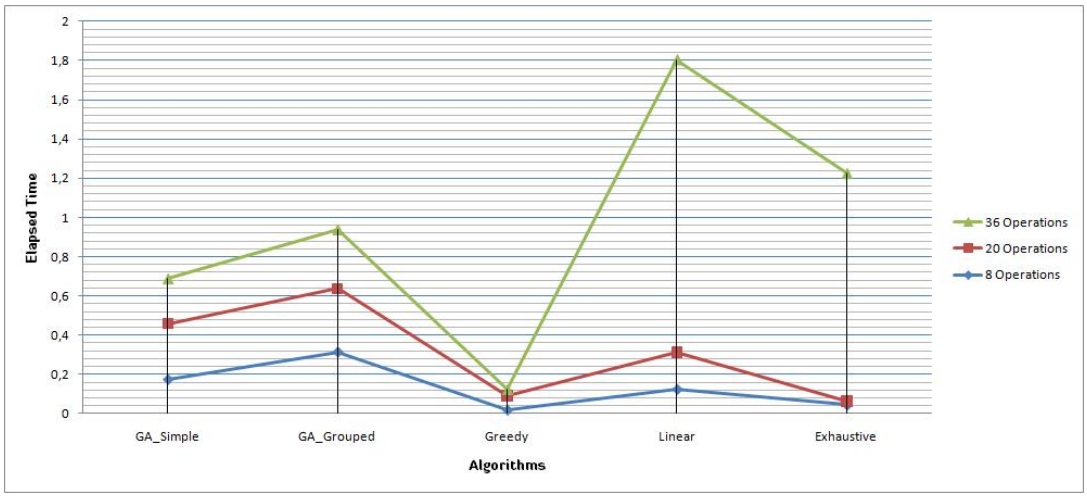

Figure 4.2: Elapsed Time Change for Algorithms

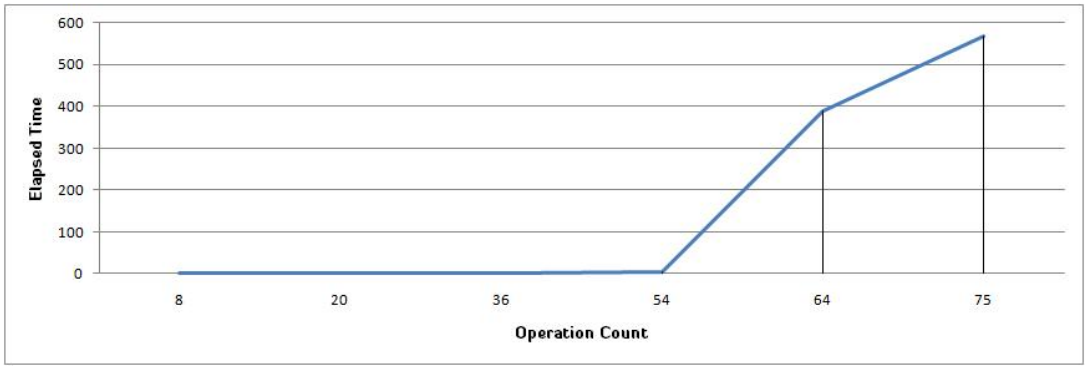

Figure 4.3: Elapsed Time for Exhaustive Search Algorithm

4.1.1 Exhaustive Search Implementation

This algorithm is used for getting reference gain values for input scenarios so none of the bounding approaches are analyzed. Results are optimum for this implementation as all operation subsets are examined. However, elapsed time values shown in Figure 4.3 is very high for the studied problem. Therefore, this implementation is not useful for real world military environment's resource allocation problem.

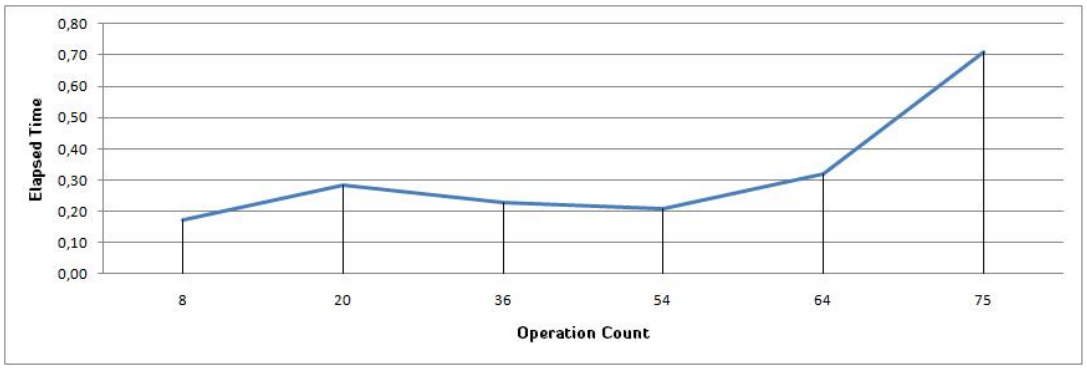

Figure 4.4: Elapsed Time for GA_Simple Algorithm

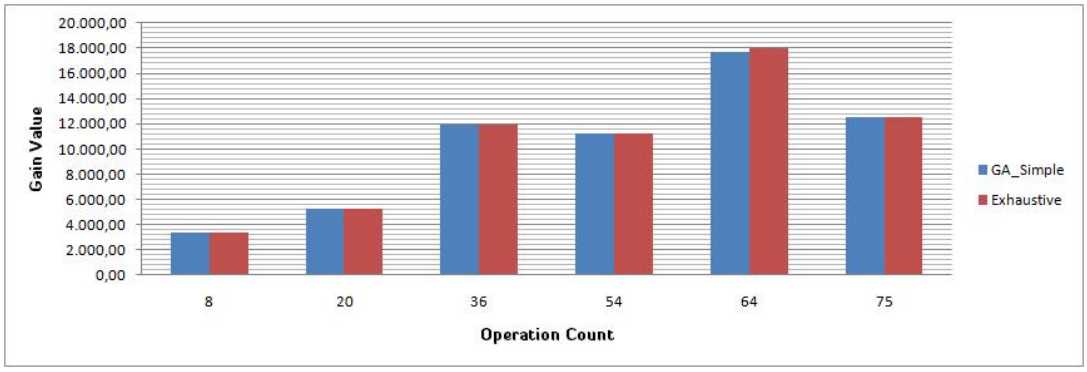

Figure 4.5: Gain Values for GA_Simple Algorithm

4.1.2 Simple Genetic Algorithm Implementation

A simple genetic algorithm (GA_Simple) with **single random point crossover** is used for collecting results. Figure 4.4 shows the efficiency analysis of this algorithm for the related input set by comparing elapsed times with corresponding scenario's operation size.

Simple genetic algorithm's elapsed time graph depicts that; this implementation can find solutions for huge sized input scenarios within a reasonable time. Figure 4.5 shows the effectivity analysis of this algorithm by comparing values with reference values (exhaustive search results) for different sized inputs.

Figure 4.5 depicts that; solutions of simple genetic algorithm implementation is equal to the reference values for most of the scenarios and difference is less than two percentage for the radical scenarios with very low capacity rates.

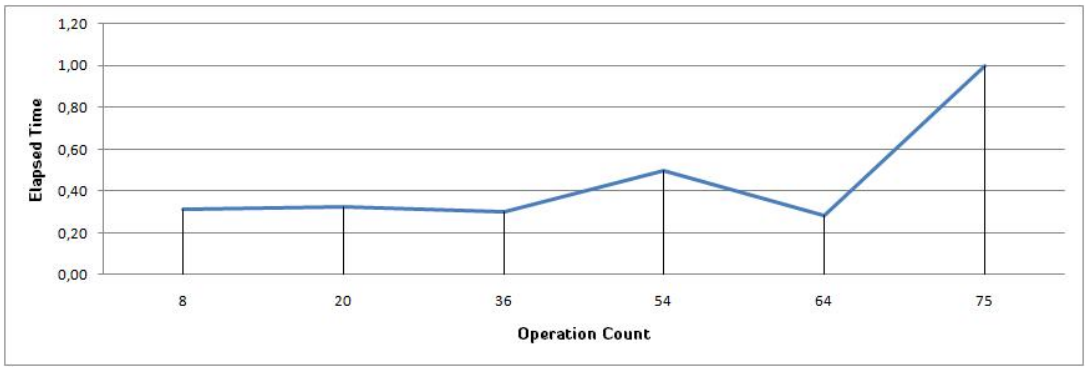

Figure 4.6: Elapsed Time for GA_Grouped Algorithm

4.1.3 Genetic Algorithm Implementation with Fleet Grouping (GA_Grouped)

This implementation is very similar to the simple genetic algorithm with a small difference on crossover methodology. In this algorithm **boundary points crossover** approach is applied to genomes which are grouped by required fleet. Solutions' gain values are equal to the simple genetic algorithm implementation, so effectivity of this implementation is shown to be same with that algorithm. In terms of elapsed time, there is a difference with simple genetic algorithm implementation, Figure 4.6 shows the efficiency analysis of this implementation.

Figure 4.6 depicts that; this implementation can find solutions for huge sized input scenarios within a reasonable time. This time is slightly larger than simple genetic algorithm implementation's elapsed time.

4.1.4 Greedy Algorithm Implementation

Scenarios in the first input set are also used for examining the greedy algorithm in terms of efficiency and effectivity. Figure 4.7 shows the efficiency analysis of this implementation.

As this algorithm tries operations of input scenario greedily, elapsed time values are very small according to other algorithm. Figure depicts that a solution can be found in a reasonable time for studied problems. Figure 4.8 shows the effectivity analysis of this algorithm by comparing values with reference values (exhaustive search results)

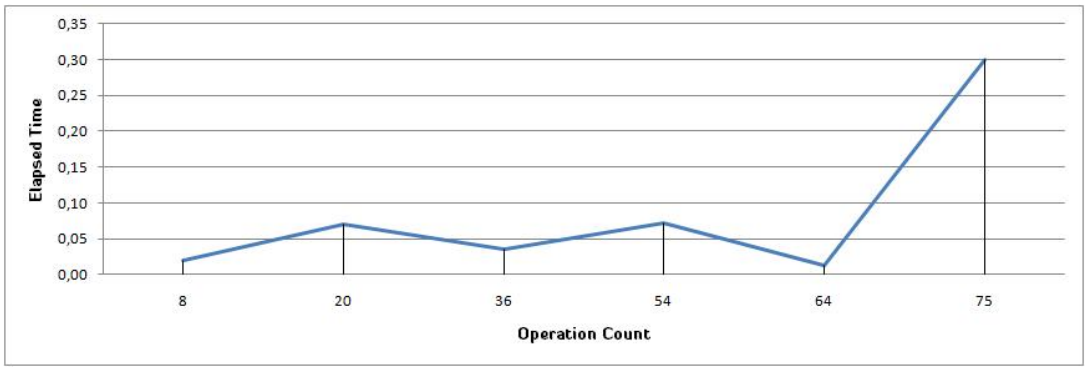

Figure 4.7: Elapsed Time for Greedy Algorithm

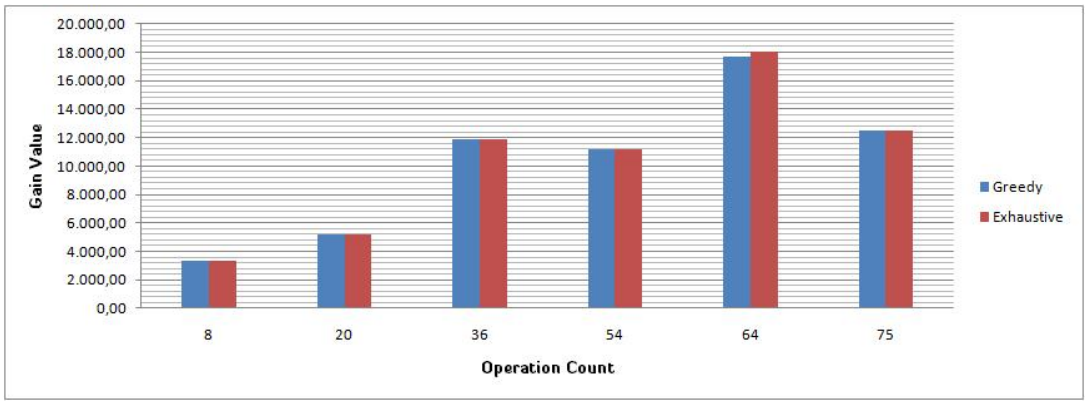

Figure 4.8: Gain Values for Greedy Algorithm

for different sized inputs.

As can be seen from the Figure 4.8; solutions of greedy algorithm implementation are generally worse than reference solutions (exhaustive search results). This difference is 10 percentages maximally for the examined input scenarios.

4.1.5 Integer Linear Programming Implementation

Scenarios in the first input set are also used for examining the integer linear programming implementation in terms of efficiency and effectivity. However, last scenario with 75 operations cannot be solved within 10000 seconds with this algorithm. Experiment with larger input size not given in the results table also supports the insolvability in short time period. Figure 4.9 shows the efficiency analysis of this implementation.

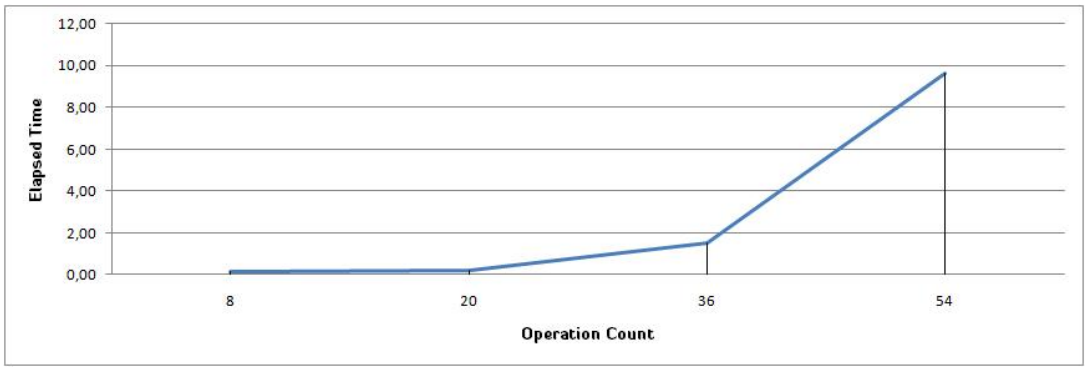

Figure 4.9: Elapsed Time for Integer Linear Programming Algorithm

Figure 4.9 depicts that; integer linear programming's elapsed time values are increasing exponentially against the linear increase in input size. Moreover, scenarios with 64 and 75 operations are not added to Figure, as these elapsed time values are huge or cannot be examined. This algorithm's effectivity is great according to applied scenarios as all of the gain values found by integer linear programmer are equal to the exhaustive search results.

4.2 Results for Input Set With High Capacity Rate

This input set contains air combat scenarios with different operation sizes. All scenarios have a capacity rate greater than one which means that all targets are able to be destroyed according to given solutions of selected algorithm implementations. Because of the high resource capacity, these input scenarios can be classified as easy optimization problems. Therefore, a preferable solver should guarantee the most effective solution.

Results of this input set mainly analyze the time complexity of genetic algorithm variations and greedy algorithm implementation. In addition, they are compared with each other according to effectivity. Figure 4.10 shows the results for the input set.

Results show us that solutions of genetic algorithm variations do not differ for each problem instance. However, greedy algorithm solutions are not as effective as genetic

INPUT SCENERIO		GA_SIMPLE		GA_GROUPED		GREEDY	
Operations	Capability	Gain	Time(s)	Gain	Time(s)	Gain	Time(s)
25	1,205	12171,62	0,50	12171,62	0,65	12171,62	0,08
47	1,393	9904,78	0,17	9904,78	0,32	9209,60	0,07
148	2,589	14051,19	1,05	14051,19	1,74	14051,19	0,02
288	1,615	26009,09	1,82	26009,09	2,94	26009,09	0,02
475	1,575	36677,38	2,58	36677,38	4,15	36677,38	0,06

Figure 4.10: Results of Input Set 2

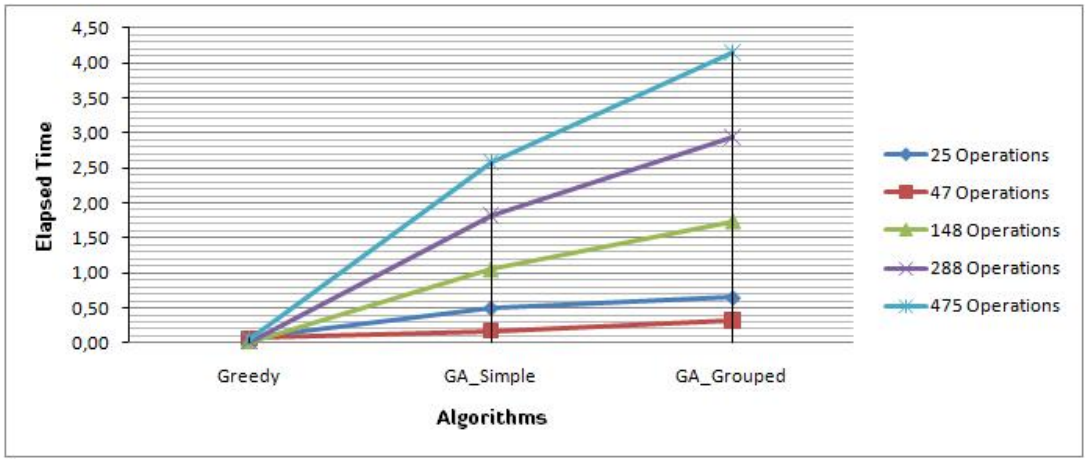

Figure 4.11: Elapsed Time Values of Algorithms

algorithms' solution in terms of gain value. Genetic algorithm variations' gain values are equal to the available maximum gain values of the input scenarios. Greedy algorithm can reach to the best solution for some scenarios, but there is always a risk of *reaching local maxima* for that implementation. That risk decreases the validity of greedy approach's fast solutions.

Elapsed time for each algorithm differs as their time complexities are not equal. Capability value seems to be unimportant for the elapsed time if it is greater than one. Figure 4.11 shows the elapsed time value changes of algorithms according to input size changes.

Elapsed time graph depicts that all problems with high capacity rate can be solved by genetic algorithm variations and greedy algorithm. Despite the effectivity problem,

INPUT SCENERIO		GA_SIMPLE		GA_LAYERED_NORMAL		GA_LAYERED_C		GA_BOUNDARY		GA_BOUNDARY_C		GREEDY	
Operations	Capability	Gain	Time(s)	Gain	Time(s)	Gain	Time(s)	Gain	Time(s)	Gain	Time(s)	Gain	Time(s)
25	0,187	4518,30	0,27	4528,60	0,47	4520,880	0,420	4528,600	0,650	4528,60	0,46	4497,69	0,11
66	0,626	12532,53	0,48	12849,01	0,71	12849,010	0,670	12849,010	1,080	12849,01	0,78	11878,73	0,10
134	0,635	15795,20	0,65	15795,20	1,35	15795,200	1,140	15795,200	1,750	15795,20	1,15	14774,36	0,10
179	0,512	19184,85	0,86	19667,43	1,54	19448,750	1,370	19523,250	1,480	19406,25	1,16	17626,18	0,11
221	0,483	27495,38	0,97	28128,32	1,80	28039,840	1,710	28039,840	1,490	28128,32	1,28	25057,31	0,09
266	0,629	28519,95	0,97	29218,28	2,11	28704,530	1,930	29177,530	2,130	29001,98	1,82	26274,28	0,10
282	0,482	29719,72	0,98	30492,74	2,04	30341,080	1,460	30408,480	2,100	30492,74	1,68	27042,53	0,11
349	0,902	29342,36	1,35	30221,17	2,87	30044,860	2,240	30044,860	3,360	29765,59	2,50	27516,13	0,10

Figure 4.12: Results of Input Set 3

greedy algorithm has the best efficiency, simple genetic algorithm has the average and genetic algorithm has the worst efficiency. However, these differences in elapsed time values are extremely small when compared to the exhaustive search and linear programming implementations.

4.3 Results for Input Set With Low Capacity Rate

This input set contains scenarios with low capacity rate (smaller than one) which means that there is not enough resource for destroying all targets. Finding solution for scenarios with these characteristics becomes harder as the gain values may vary much according to the selected solution. Greedy algorithm and genetic algorithm variations are examined with this input set in order to analyze the effectivity differences. Greedy algorithm has still effectivity risks as in the input set with high capacity rate. In addition, exhaustive search and integer linear programming implementations are shown to have unacceptable execution times for problems with medium or large input size. Therefore, different sized scenarios with low capacity rate are run only with greedy algorithm and genetic algorithm customizations which are listed in Table 4.1.

Figure 4.12 shows the results of these algorithms for the eight different sized scenarios in the input set.

These results are examined in terms of effectivity for this input set. Greedy algorithm results are much worse than genetic algorithms. Moreover, custom genetic algorithm implementations are very efficient as it takes less than 4 seconds in the worst case

Table 4.1: Examined Genetic Algorithm Customizations

Genetic Algorithm	**Properties**
Simple Implementation (GA_Simple)	Simple Maximum Gain Encoding Single Point Random Crossover Simple Objective Based Fitness Function
Layered Crossover Implementation (GA_Layered_Normal)	Fleet Based Maximum Gain Encoding Layered Hybrid Crossover Simple Objective Based Fitness Function
Layered Crossover Implementation with Custom Fitness (GA_Layered_C)	Fleet Based Maximum Gain Encoding Layered Hybrid Crossover Percentage Based Custom Fitness Function
Boundary Crossover Implementation (GA_Boundary_Normal)	Fleet Based Maximum Gain Encoding Boundary Points Crossover Simple Objective Based Fitness Function
Boundary Crossover Implementation with Custom Fitness (GA_Boundary_C)	Fleet Based Maximum Gain Encoding Boundary Points Crossover Percentage Based Custom Fitness Function

of this experiment. In order to examine effectivity rates, we calculated the result performance as a percentage value by comparing each gain value with the best gain value of the same scenario. Figure 4.13 shows the result performance percentages of all implementations according to the best gain value achieved.

From the analyzed solution performances, following conclusions can be drawn for algorithm implementations:

- Customization with name **GA_Layered_Normal** is the most effective implementation for the given input scenarios with low capacity rate.
- Simple genetic algorithm implementation **(GA_Simple)** performs worse than all other genetic algorithm customizations.
- Greedy algorithm implementation **(Greedy)** performs worse than all other algorithms.

Figure 4.13: Result Performance of Genetic Algorithm Customizations

- Boundary points crossover based implementations' performance is less or equal to the performance of layered hybrid crossover implementations.
- Percentage based custom fitness function decreases the effectivity performance for the layered hybrid crossover implementation.
- Percentage based custom fitness function's effect on boundary points crossover cannot be analyzed according to gain values. Because **GA_Boundary_C** performs better than **GA_Boundary_Normal** for some scenarios and worse for some others.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusions

In this thesis, we analyzed the resource allocation problem of military air forces during ATO production. Then, we formulated problem as a combinatorial optimization problem and represented the problem in a way that all candidate algorithms can be realized.

Genetic, integer programming, greedy and exhaustive search algorithms are developed for this problem. In addition, specific features and customizations are made to those implementations. A plug-in based framework is used as it supports new algorithm addition for the comparison. Each implementation is examined in terms of effectivity and efficiency in order to optimize the basic requirements of resource allocation problem instance. Three different input sets are created for different algorithm evaluations. Each air combat scenario is solved by selected ones from all implemented algorithms. Finally, solutions are collected from results and their comparable values, such as gain and elapsed time, are displayed.

In terms of effectivity, exhaustive search and integer linear programming solver are the implementations that give the optimized gain value for the air combat scenario. However, time complexities of these algorithms are huge and required time for medium sized scenario is unacceptable for a mission critical job on a combat. Therefore, those algorithms are not applicable for real world military environment despite their high effectivity.

According to experiments made for this research, greedy algorithm requires the least elapsed time for execution because of the lowest complexity. On the other hand, this efficient algorithm does not bother on the solution effectivity as much as other implementations. Gain values of this algorithm's solutions are generally worse than all other implemented algorithms. Greedy algorithm would not be applicable to real world military environment because being efficient is not enough if the air force lose gain from the air attack.

Linear programming solver is one of the added algorithm implementations for resource allocation problem. As ATO preparation can be only made by integer values, integer linear programming implementation is used for the comparison. High performance of linear programming in combinatorial optimization problems come with optimized gain values which are equal to the reference value created by exhaustive search. However, efficiency problem is encountered with increasing number of integer valued variables. Time elapsed for this algorithm's execution can exceed 1 day for medium sized air combat scenarios. So this implementation cannot be used in real world military environment.

After combining all experiment values, we can say that genetic algorithm is practically best solution provider for studied problem. Solutions are created nearly as quickly as greedy algorithm (smaller than 5 seconds for all input scenarios) and gain values of those solutions are not much worse than reference values created by exhaustive search. Gain value difference does not exceed 1 percentage for all input scenarios. During our work, we implemented different approaches of genetic algorithm terms like encoding, crossover and fitness.

Experiments with different genetic algorithm implementations let us conclude that; dividing genomes into independent sub-genomes according to a common feature or a common interest will improve the objective function value. Encoding and crossover approaches should also be selected accordingly. Sub-genomes are determined by the required fleet identifier so each sub-genome related with a unique fleet is not dependent on another sub-genome. However, bits in a sub-genome are strongly dependent on another. Selection and crossover mechanism of a better genetic algorithm implementation should be based on such sub-genomes. In our problem, algorithm

implementation with layered hybrid crossover performs best. Because, that crossover approaches both use sub-genome based attribute heritage and selection mechanisms.

We can conclude that; genetic algorithm implementations are well suited for combinatorial optimization problems with high efficiency and effectivity. Moreover, dividing a genetic problem into sub-problems with problem specific knowledge may improve the effectivity of solutions.

5.2 Future Work

Current study is based on binary encoding of genetic algorithm but there are optimization problem implementations with real number encoding and they are proven to work more effectively. Useless spaces required for binary encoding will be also removed and space complexity will also decrease for such implementation.

Crossover from varying number of points is not examined throughout this work. If that point count and positions are selected by a guiding dynamic fitness function, attribute heritage to off-springs will improve. This dynamic fitness function will provide a convergence controlled variation as it will support population areas with higher convergence.

Throughout this thesis, steady state selection, which is proven to improve effectivity by previous study [8] is applied to the populations. I believe that performance will improve if problem specific features are used throughout the selection process.

REFERENCES

[1] F. S. Hillier, G. J. Lieberman, F. Hillier, and G. Lieberman, *MP Introduction to Operations Research*. McGraw-Hill Science/Engineering/Math, 2004.

[2] J. P. Sowa, "The computer assisted air tasking order preparation system, an enhancement to the computer assisted force management system (cafms) and constant watch programs," Military Report in The Defense Technical Information Center, 1981.

[3] M. Kress, *Operational Logistics: The Art and Science of Sustaining Military Operations*. Springer, 2002.

[4] A. Schrijver, "A course in combinatorial optimization," CWI and University of Amsterdam, 2006.

[5] J. M. Rosenberger, H. S. Hwang, R. P. Pallerla, A. Yucel, R. L. Wilson, and E. G. Brungardt, "The generalized weapon target assignment problem," 10th Int. Conf. Com. and Cont. Res. and Techn. Symp, 2005.

[6] V. C. Li, G. L. Curry, and E. A. Boyd, "Towards the real time solution of strike force asset allocation problems," *Computers & Operations Research*, vol. 31, no. 2, pp. 273–291, 2004.

[7] Z. J. Lee, S. F. Su, and C. Lee, "A genetic algorithm with domain knowledge for weapon target assignment problems," *Journal of the Chinese Institute of Engineers*, vol. 45, pp. 287–295, 2002.

[8] S. Tikveş, "Design and implementation of an asset target allocation system for air tasking orders," M.Sc. Thesis in Computer Engineering Department of Hacettepe University, 2007.

[9] L. J. and A. Iii, "Joint publication 3-30 command and control for joint air operations," 2003.

[10] D. Aberdeen, S. Thiébaux, and L. Zhang, "Decision-theoretic military operations planning," *Proceedings of the 14th International Conference on Automated Planning and Scheduling*, pp. 402–412, 2004.

[11] M.-C. Su, S.-C. Lai, D.-Y. Huang, L.-F. You, Y.-K. Liao, and J.-M. Huang, "A pso-based decision aid for multi-aircraft combat situations," *International Journal of Fuzzy Systems*, 2008.

[12] A. Toet and H. d. Waard, "The weapon-target assignment problem," CALMA Report CALMA.TNO.WP31.AT.95c, 1995.

[13] R. K. Ahuja, A. Kumar, K. C. Jha, and J. B. Orlin, "Exact and heuristic algorithms for the weapon-target assignment problem," *Operations Research*, vol. 55, no. 6, pp. 1136–1146, 2007.

[14] B. J. Griggs, G. S. Parnell, and L. J. Lehmkuhl, "An air mission planning algorithm using decision analysis and mixed integer programming," *Operations Research*, vol. 45, pp. 662–676, 1997.

[15] Z. J. Lee, S. F. Su, and C. Lee, "Efficiently solving general weapon-target assignment problem by genetic algorithms with greedy eugenics," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 33, pp. 113–121, 2003.

[16] J. R. McDonnell, N. Gizzi, and S. J. Louis, "Strike force asset allocation using genetic search," *International Conference on Artificial Intelligence*, pp. 897–901, 2002.

[17] S. J. Louis, J. R. McDonnell, and N. Gizzi, "Dynamic strike force asset allocation using genetic search and case-based reasoning," 2002.

[18] L. J. Eshelman, "The chc adaptive search algorithm," in *Foudations of Genetic Algorithms*, G. Rawlins, Ed. Morgan Kaufmann, 1990, pp. 265–283.

[19] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990.

[20] M. A. Schulze, "Linear programming for optimization," Perceptive Scientific Instruments, Inc, 1998.

[21] G. Dantzig, *Linear Programming and Extensions*. Princeton University Press, 1963. [Online]. Available: http://www.worldcat.org/isbn/0691059136

[22] I. Maros, *Computational Techniques of the Simplex Method*. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[23] N. Meggido, "Pathways to the optimal set in linear programming," *Progress in Mathematical Programming Interior-point and related methods*, pp. 131–158, 1988.

[24] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*. New York, NY, USA: John Wiley & Sons, Inc., 2000.

[25] J. Gottlieb and G. R. Raidl, *Evolutionary Computation in Combinatorial Optimization, 4th European Conference, EvoCOP 2004, Coimbra, Portugal, April 5-7, 2004, Proceedings*, ser. Lecture Notes in Computer Science. Springer, 2004, vol. 3004.

[26] J. H. Holland, *Adaptation in natural and artificial systems*. Cambridge, MA, USA: MIT Press, 1992.

[27] L. J. Eshelman and J. D. Schaffer, "Real-coded genetic algorithms and interval-schemata." *Foundation of Genetic Algorithms 2*, pp. 187–202, 1993.

[28] T. N. Bui and B. R. Moon, "On multi-dimensional encoding/crossover," in *Proceedings of the 6th International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 49–56.

[29] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms—i: representation," *International Journal of Computers and Industrial Engineering*, vol. 30, no. 4, pp. 983–997, 1996.

[30] I. Rechenberg, *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog, 1973.

[31] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[32] L. J. Eshelman, K. E. Mathias, and J. D. Schaffer, "Convergence controlled variation," *Foundation of Genetic Algorithms*, pp. 203–224, 1996.

[33] A. W. Black, "Complexity theory and np-completeness," Coventry (Lanchester) Polytechnic Dept of Computer Science, 1984.

[34] M. Sipser, *Introduction to the Theory of Computation*. International Thomson Publishing, 1996.

[35] M. G. Lagoudakis, "The 0-1 knapsack problem – an introductory survey," 1996.