ALGORITHMS FOR THE WEAPON - TARGET ALLOCATION PROBLEM

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

BY

AYŞE TURAN

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

JULY 2012

Approval of the thesis:

ALGORITHMS FOR THE WEAPON - TARGET ALLOCATION PROBLEM

submitted by **AYŞE TURAN** 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. Adnan Yazıcı _____
Supervisor, **Computer Engineering Dept., METU**

Assoc. Prof. Dr. Halit Oğuztüzün _____
Co-supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. İ. Hakkı Toroslu
Computer Engineering Dept., METU _____

Prof. Dr. Adnan Yazıcı
Computer Engineering Dept., METU _____

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

Asst. Prof. Dr. Selim Temizer
Computer Engineering Dept., METU _____

Lead Software Design Eng. Dilek Arslan
Software Engineering Dept., ASELSAN 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: AYŞE TURAN

Signature :

ABSTRACT

ALGORITHMS FOR THE WEAPON - TARGET ALLOCATION PROBLEM

Turan, Ayşe
M.Sc., Department of Computer Engineering
Supervisor : Prof. Dr. Adnan Yazıcı
Co-Supervisor : Assoc. Prof. Dr. Halit Oğuztüzün

July 2012, 119 pages

Within the air defense domain, the Weapon-Target Allocation problem is a fundamental problem. This problem deals with the allocation of a set of firing units or weapons to a set of hostile targets so that the total expected effect on targets is maximized. The Weapon-Target Allocation problem has been proven to be NP-Complete by Lloyd and Witsenhausen [14].

In this thesis, the use of various algorithms including *search algorithms*, *maximum marginal return algorithms*, *evolutionary algorithms* and *bipartite graph matching algorithms* are demonstrated to solve the problem. Algorithms from the literature are adjusted to the problem and implemented. In addition, existing algorithms are improved by taking care of the maximum allowed time criterion. A testbed is developed to be able to compare the algorithms. The developed testbed allows users to implement new algorithms and compare the algorithms that are selected by the users easily. Using the testbed, implemented algorithms are compared based on optimality and performance criteria. The results are examined and by combining the algorithms that give better results, a

new algorithm is proposed to solve the problem more efficiently. The proposed algorithm is also compared to the other algorithms and computational results of the algorithms are presented.

Keywords: Weapon-Target Allocation, Assignment, Air Defense, Optimization, Performance Evaluation

ÖZ

SİLAHLARIN HEDEFLERE TAHSİSİ PROBLEMİNE YÖNELİK ALGORİTMALAR

Turan, Ayşe
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Adnan Yazıcı
Ortak Tez Yöneticisi : Doç. Dr. Halit Oğuztüzün

Temmuz 2012, 119 sayfa

Hava Savunma alanında, silahların hedeflere tahsis edilmesi bilinen temel bir problemdir. Bu problem, sistemde mevcut bir grup silahın bir grup düşman hedefe, hedeflerde meydana gelecek hasarı maksimize edecek şekilde tahsis edilmesini ele alır. Silahların hedeflere tahsisi probleminin NP-Tam olduğu Lloyd ve Witsenhausen tarafından ispatlanmıştır [14].

Bu tezde, problemin çözümüne yönelik arama algoritmaları, maksimum marjinal getiri algoritmaları, evrimsel algoritmalar ve ikili grafik eşleme algoritmalarını da içeren pek çok algoritma açıklanmıştır. Literatürde var olan algoritmalar probleme uyarlanarak gerçeklenmiştir. Ek olarak, var olan algoritmalar, algoritmanın izin verilen maksimum çalışma zamanı da göz önünde bulundurulacak şekilde iyileştirilmiştir. Algoritmaları karşılaştırabilmek için bir test ortamı geliştirilmiştir. Geliştirilen test ortamı, kullanıcıların kolaylıkla yeni algoritmalar gerçekleyerek, seçtiği algoritmaları karşılaştırabilmesine imkan sağlamaktadır. Geliştirilen test ortamı kullanılarak gerçeklenen algoritmalar optimum çözüme

yakınlık ve performans açısından karşılaştırılmıştır. Sonuçlar incelenerek daha iyi sonuç veren algoritmalar seçilmiş ve bu algoritmalar birleştirilerek yeni bir algoritma önerilmiştir. Önerilen algoritma da diğer algoritmalarla karşılaştırılmış ve elde edilen sonuçlar sunulmuştur.

Anahtar Kelimeler: Silahların Hedeflere Tahsisi, Hava Savunma, Optimizasyon, Performans Değerlendirmesi

To My Parents

ACKNOWLEDGMENTS

First of all, I would like to express my sincere thanks to my advisors Prof. Dr. Adnan Yazıcı and Assoc. Prof. Dr. Halit Oğuztüzün for their support, understanding and guidance throughout this thesis work.

I would like to thank ASELSAN Inc. for facilities provided for the completion of this thesis and TÜBİTAK BİDEB for the financial support.

I am also thankful to my colleague Dilek Arslan for her support, consideration and valuable comments and suggestion for this work.

I am greatly indebted to my family for their encouragements, unconditional support and patience.

Lastly, I would like to thank Murat for always being there for me.

TABLE OF CONTENTS

ABSTRACT iv

ÖZ vi

ACKNOWLEDGMENTS ix

TABLE OF CONTENTS x

LIST OF TABLES xiii

LIST OF FIGURES xiv

LIST OF ABBREVIATIONS xvi

CHAPTERS

1 INTRODUCTION 1

1.1 Background 1

1.1.1 Air Defence 1

1.1.2 Threat Evaluation and Weapon Allocation System 2

1.1.3 A Ground-Based Air Defense Scenario 2

1.2 Scope and Objectives 10

1.3 Contributions of the Thesis 11

1.4 Organization of the Thesis 11

2 WEAPON - TARGET ALLOCATION PROBLEM 12

2.1 Problem Formulation 12

2.2 Related Work on Weapon - Target Allocation Problem . 16

3 ALGORITHMS FOR WEAPON - TARGET ALLOCATION PROBLEM 19

3.1 Representations 19

3.2 Search Algorithms 22

3.2.1 Exhaustive Search Algorithm 22
3.2.2 Random Search Algorithm 26
3.3 Maximum Marginal Return (MMR) Algorithms 27
3.3.1 Greedy MMR Algorithm 27
3.3.2 Greedy MMR Algorithm Improved with Local Search 29
3.3.3 Random MMR Algorithm 29
3.3.4 Advanced MMR Algorithm 31
3.3.5 Advanced MMR Algorithm Improved with Local Search 31
3.4 Evolutionary Algorithms 33
3.4.1 Genetic Algorithm 38
3.4.2 Genetic Algorithm Improved with MMR Algorithms 41
3.4.3 Ant-Colony Optimization Algorithm 44
3.4.4 Particle Swarm Optimization Algorithm 52
3.4.5 Particle Swarm Optimization Algorithm Improved with MMR Algorithms 57
3.5 Bipartite Graph Matching Algorithms 57
3.5.1 Munkres' Assignment Algorithm 61
3.5.2 Munkres Assignment Algorithm Improved with MMR Algorithms 75
3.5.3 Advanced Munkres Assignment Algorithm . . 75
3.6 A Suggested Algorithm 76
4 EVALUATION OF ALGORITHMS 77
4.1 Testbed Developed for the Evaluation of Algorithms . . 78
4.1.1 Requirements of the Testbed 78
4.1.2 Design Model of the Testbed 79
4.2 Experimental Results 88
4.2.1 Performance Evaluation 88
4.2.2 Optimality Evaluation 90
4.2.2.1 Objective Function Value Comparison 91

4.2.2.2 Deviation Comparison 93
4.2.2.3 Rank Comparison 93
5 CONCLUSION AND FUTURE WORK 98
REFERENCES 100
APPENDICES
A COMPUTATIONAL RESULTS OF THE EXPERIMENTS . . . 102

LIST OF TABLES

TABLES

Table A.1 Execution Times (in ms) - 1 104
Table A.2 Execution Times (in ms) - Cont'd 105
Table A.3 Execution Times (in ms) - Cont'd 106
Table A.4 Execution Times (in ms) - Cont'd 107
Table A.5 Objective Function Values - 1 108
Table A.6 Objective Function Values - Cont'd 109
Table A.7 Objective Function Values - Cont'd 110
Table A.8 Objective Function Values - Cont'd 111
Table A.9 Deviations 112
Table A.10Deviations - Cont'd 113
Table A.11Deviations - Cont'd 114
Table A.12Deviations - Cont'd 115
Table A.13Ranks 116
Table A.14Ranks - Cont'd 117
Table A.15Ranks - Cont'd 118
Table A.16Ranks - Cont'd 119

LIST OF FIGURES

FIGURES

Figure 1.1 Typical Air Defense Scenario 2

Figure 1.2 Closest Point of Approach 5

Figure 3.1 Visiting Orders of Nodes in Breadth-First Search 23

Figure 3.2 Visiting Orders of Nodes in Depth-First Search 23

Figure 3.3 Flow Chart of Evolutionary Algorithms 34

Figure 3.4 Roulette-Wheel Selection 36

Figure 3.5 Stochastic Universal Sampling 36

Figure 3.6 Graph Representation of Weapon - Target Allocation Problem 45

Figure 3.7 Flow Chart of Particle Swarm Optimization Algorithm . . . 54

Figure 4.1 Classes and Packages of Algorithm Package 80

Figure 4.2 Search Algorithms Package 80

Figure 4.3 Maximum Marginal Return Algorithms Package 81

Figure 4.4 Evolutionary Algorithms Package 82

Figure 4.5 Bipartite Graph Matching Algorithms Package 83

Figure 4.6 Suggested Algorithm Package 83

Figure 4.7 Scenario Package 85

Figure 4.8 Main Screen 86

Figure 4.9 Execution Times - Small Scale Problem Instance 89

Figure 4.10 Execution Times - Large Scale Problem Instance 89

Figure 4.11 Execution Times - Average 90

Figure 4.12 Objective Function Values - Small Scale Problem Instance . . 91

Figure 4.13 Objective Function Values - Large Scale Problem Instance . . 92
Figure 4.14 Objective Function Values - Average 92
Figure 4.15 Deviations - Small Scale Problem Instance 93
Figure 4.16 Deviations - Large Scale Problem Instance 94
Figure 4.17 Deviations - Average 94
Figure 4.18 Ranks - Small Scale Problem Instance 95
Figure 4.19 Ranks - Large Scale Problem Instance 96
Figure 4.20 Ranks - Average 96
Figure 4.21 Ranks - Algorithm Categories 97

LIST OF ABBREVIATIONS

CPAIUOT	Closest Point of Approach in Units of Time
CPA	Closest Point of Approach
GA	Genetic Algorithm
IFF	Identification Friend of Foe
MMR	Maximum Marginal Return
NATO	North Atlantic Treaty Organization
NP	Nondeterministic Polynomial Time
TBH	Time Before Hit
TE	Threat Evaluation
TEWA	Threat Evaluation and Weapon Allocation
VLSN	Very Large Scale Neighborhood Search
WA	Weapon Allocation
WTA	Weapon - Target Assignment

CHAPTER 1

INTRODUCTION

1.1 Background

1.1.1 Air Defence

Air defence is defined as *"all measures designed to nullify or reduce the effectiveness of hostile air action"* in NATO Glossary of Terms and Definitions, AAP-6 [2].

The concept of air defence has appeared soon after man took to the air and airplanes were used in attacks. With the impact of technology on warfare, both aircraft and air defence weapons have been developed and improved. Today, many kinds of airplanes such as fixed-wing or rotary-wing aircraft, unmanned aerial vehicles and missiles are used in attack, and many kinds of weapons such as rockets and artilleries are used in air defence.

Basically, an air defence system is composed of radars detecting hostile aircraft, defensive weapons firing out the aircraft detected by radars and command control units making the decisions. Although, the technology behind the development of radars and weapons is very important, effective usage of the weapons against aerial threats is much more crucial. Operators or the decision makers have to evaluate the tactical situation in real time and protect assets that have no defense capability against aerial threats or enemy targets by assigning available weapons to them. Since the decisions should be made in real time, responsibilities are typically divided between a number of operators and computerized

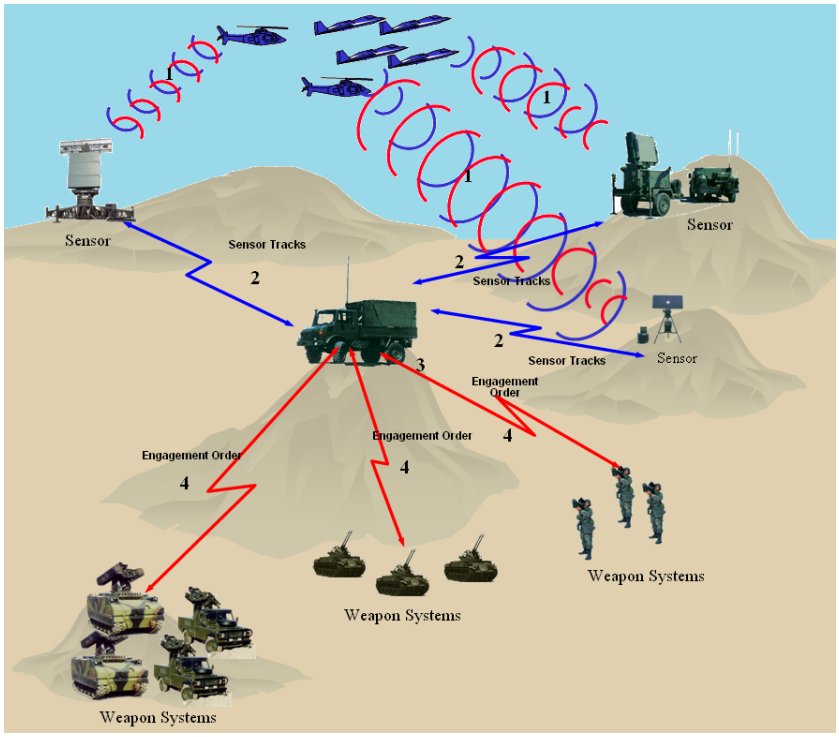

Figure 1.1: Typical Air Defense Scenario

systems that help the operators during the decision making processes.

1.1.2 Threat Evaluation and Weapon Allocation System

Threat Evaluation and Weapon Allocation (TEWA) system is a computerized air defense system that assigns threat values to aerial threats in real time (Threat Evaluation) and uses these threat values to propose possible engagements of aerial threats by weapons (Weapon Allocation).

Both Threat Evaluation (TE) and Weapon Allocation (WA) processes are crucial since the outcome of these processes directly affects military resources and human life.

1.1.3 A Ground-Based Air Defense Scenario

A typical ground-based air defense scenario is illustrated in Figure 1.1.

It contains 4 processes:

1. **Sensing Process:**

   In the sensing process, various sensors in the system observe the tactical environment, detect aerial targets, and assign those targets an identification, *i.e. friend, assumed friend, neutral, suspect or hostile*. Then, sensor tracks containing information about both hostile and suspect targets are emitted from the sensors. This process is marked as 1 and 2 in Figure 1.1.

   The capabilities and properties of targets such as type, position, kinematics, radius of operation need to be defined for the purpose of evaluating the threat values posed by targets to defended assets. These parameters are defined in the sensing process. In this process, sensors detect aerial targets and identify them.

   The accuracy and quality of the target information is crucial and directly depend on the choice of sensors. There are various kinds of sensors used in military. These sensors are usually classified as *short range* (0 - 50 km), *medium range* (50 - 200 km) and *long range* (> 200 km).

   The sensing process is not dealt with in this thesis, but it is direcly related to threat evaluation process.

2. **Target Management Process:**

   In the target management process, sensor targets coming from various sensors are combined into a single system target by information fusion techniques to set up an aerial picture. This process is also known as *Fusion Phase* and marked as 3 in Figure 1.1.

   When several sensors exist in the system, it is quite possible that the sensors detect the same target. The combination of targets coming from various sensors into a single target is done using information fusion methods. The fusion method used in the target management process is crucial and stated as *"the fusion method must be designed carefully, because an inappropriate fuser can render the system worse than the worst individual sensor."* [17].

The information fusion method used in the target management process is not the main focus of this thesis. But, it is an important process and the output produced from this process is the input to the threat evaluation process.

3. **TE Process:**

   In the TE process, system targets are evaluated to measure the threatening behavior of a target with respect to defended assets by threat evaluation techniques for each target, and a target value is calculated. This process is marked as 3 in Figure 1.1.

   According to Paradis et al. *"In the defense domain, TE refers to the part of threat analysis concerned with the ongoing process of determining if an entity intends to inflict evil, injury, or damage to the defending forces and its interests, along with the ranking of such entities according to the level of threat they pose"* [3].

   In the TE process, a threat value (between 0 and 1) is calculated for each target and defended asset pair. Then, a prioritized threat list, going from the most dangerous threat to the least is constructed from these threat values.

   The identification and prioritization of threats is very important because errors such as incorrectly identifying a non-threat as a threat or prioritizing a lesser threat as a greater threat can result in engaging the wrong target that might have severe consequences. An actual example of this situation occurred in 1988: The USS Vincennes (CG-49) ship shot down Iran Air Flight 655 over the Persian Gulf and killed all 290 civilian passengers onboard, including 38 non-Iranians and 66 children just because it has misidentified the Iranian Air-bus as an attacking F-14 Tomcat fighter [4].

   In order to evaluate the threat values for each target and defended asset pair, a large number of parameters, many of which are related to each other, have been suggested in the literature. After an exhaustive literature survey of publications dealing with TE in the information fusion domain and related areas, Johansson et al. has classified these parameters into three categories [5]:

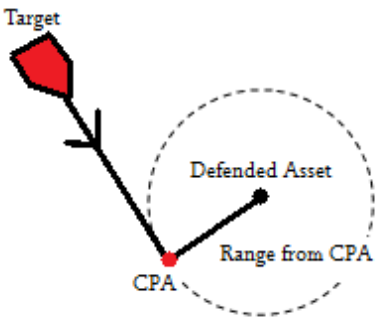

Figure 1.2: Closest Point of Approach

(a) **Proximity Parameters**

Proximity parameters are parameters that measure the target's proximity to the defended asset.

- **Range from Closest Point of Approach (CPA):** CPA is the point where the target eventually will be the closest to the defended asset. It is the orthogonal projection of the position of the defended asset on the extension of the target's velocity vector. The range from CPA is the distance between the position of the defended asset and the CPA. CPA and range from CPA is shown in Figure 1.2. The larger the range from CPA, the lower the threat value.
- **Time to CPA (TCPA):** The range from CPA divided by target speed. The larger the time to CPA, the lower the threat value.
- **CPA in Units of Time (CPAIUOT):** Time it would take the target to hit the defended asset after a 90 degree turn at its CPA (range from CPA divided by target speed). The larger the CPAIUOT, the lower the threat value.
- **Time Before Hit (TBH):** Time it would take the target to hit the defended asset (Eqn. 1.1). The larger the TBH, the lower the threat value.

$$TBH = TCPA + CPAIUOT \tag{1.1}$$

- **Distance:** The distance between the target and the defended

asset. The larger the distance, the lower the threat value.

(b) **Capability Parameters**

Capability parameters are parameters that measure the target's capability to threaten the defended asset (lethality of the target).

- **Target Type:** The type of the enemy aircraft which can be identified from other parameters such as answer to IFF-interrogation, electronic support measures, speed, etc.
- **Weapon Type:** The armament used by the enemy aircraft during the engagement of defended assets. It can be inferred from the type of the target.
- **Fuel Capacity:** The fuel capacity of the enemy aircraft. It can be inferred from the type of the target. The lower the fuel capacity of the target, the lower the threat value.
- **Maximum Radius of Operation:** The maximum radius of operation of the target can be inferred from the fuel capacity of the target. The lower the target's maximum radius of operation, the lower the threat value.

(c) **Intent Parameters**

Intent parameters are parameters that measure the intent of the target. Unlike capability, intent is generally more difficult to assess because it is more subjective.

- **Target's Kinematics:** Target's velocity in combination of its altitude.
- **Number of Recent Maneuvers:** The more the target maneuvers, the larger the threat value.

Yet, it is possible to increase the number of parameters that will affect the value of the threat and it may be very difficult to measure or define. TE is a complex task accomplished by a team of highly skilled personnel. It requires domain knowledge and mental integration of data from many sources. That integration requires a high level of tactical expertise, including knowledge of the types of threats, defended assets' and weapon

systems' missions, doctrine, and assessment heuristics built from experience [6]. Hence, TE process is a poorly defined process because it requires the reproduction of operator thought in real time and includes many parameters that may not be measurable easily.

Due to the complex thought processes and cumulative experience embodied in a human operator, a TE system that is capable of replacing or outperforming a human operator is currently impossible. So, the focus of a TE system should be to operate as a Decision Support System providing TE results and various derived threat attributes to a ground-based air defense operator that are typically too tedious to compute by hand [8].

Algorithms for TE Process

Since TE process is a poorly defined process, work conducted on TE is not well-documented; only a few algorithms for TE are available in the literature.

A rule-based algorithm is suggested by using a series of experiments with U.S. Navy officers [6], [7]. Before the suggested algorithm was developed, the authors had made a questionnaire and collected data from 9 experienced U.S. Navy personnel. Then, they had decided on the parameters that would be taken into account during the threat value calculation phase, based on the collected data. Then, data had been incorporated into a rule-based algorithm.

Another kind of rule-based algorithm is suggested, in which fuzzy inference rules are used to calculate the values of threats, using altitude, speed, CPA, and range as parameters [9]. Although, the rules are simple, since the number of parameters used during the threat value calculation phase is reduced, it does not offer a sufficient solution.

In [10], a Bayesian Network based algorithm is suggested. In the algorithm the target state estimates are used during the threat value calculation phase. Evasive maneuvers, fire control radar, countermeasures, political climate, IFF squawking, flight plan agreement, platform type, and imminence are used as parameters.

In [5], another kind of Bayesian Network based algorithm is suggested. In

this algorithm, target type, weapon range, speed, TBH and distance are used as parameters and these parameters form the nodes of the Bayesian Network. In the paper, it is suggested that more nodes should be added to the Bayesian Network as a future work.

Although, the Bayesian-Network based algorithm given in [5] and the rule-based algorithm given in [9] are compared in [11], a general comparison about the performance and correctness of the algorithms are missing in the literature.

Also, in almost all of the suggested algorithms, the investigation on whether the system's calculated threat values on realistic scenarios agrees with human experts on air defense or not is missing.

Since Threat Evaluation process is a poorly defined process and requires high level of domain knowledge, it is not the main focus of this thesis. But, it is an important process and the output produced from this process is directly used in Weapon Allocation Process, which is the main focus of this thesis.

4. **WA Process:**

   In the WA process, the allocation of weapons to targets is decided based on the target values and the probability of a successful engagement. This process is marked as 4 in Figure 1.1.

   This process is also known as *Weapon Target Assignment (WTA)* or *Weapon Allocation (WA).*

   According to Paradis et al. *"In the defense domain, WA refers to the reactive assignment of weapon systems to engage or counter identified threats"* [3].

   WA process is the process of optimally assigning weapons to targets so that

   (a) the total expected survival value of the targets after all engagements is minimum or

   (b) the damage to the defended assets is minimum or

(c) the overall survivability of assets is maximum [12].

The result of a WA process may be a list of all weapon systems that are available for assignment, a list of all weapon systems that are suitable for assignment to each particular threat, a prioritized list of suitable weapon systems for each particular threat or reassignment options for substituting the assigned weapon systems with other weapon systems if required [3].

The WA problem may be considered from a number of different perspectives:

- **From the single platform perspective:** WA refers to a single platform protecting itself from threats, where assignment relates to selecting the most suitable weapon to counter a threat.
- **From the force coordination perspective:** WA refers to a command and control platform, where assignment relates to identifying the most suitably armed platform to engage or counter a threat.
- **From the threat-by-threat perspective:** WA refers to the assignment of weapon systems sequentially, in such a way that the best weapon system is essentially assigned to each threat in turn from the highest priority to the lowest priority. It is usually some type of greedy algorithm in principle.
- **From the multi-threat perspective:** WA refers to the assignment of weapon systems to the current set of threats concurrently, so that the assignment is best in some overall sense. It involves the optimization of a given objective function.

The assignment of weapons to targets in an efficient way is very important since the problem must be solved in real time. The enormous combinatorial complexity (even if there are 10 weapons and 10 targets, there are 100 combinations) of the problem implies that optimal solutions cannot be obtained in real time. Optimal solutions are applicable only when the input size is very small since the computation time of any optimal algorithm for the problem grows exponentially with the size of the problem. However, the WA problem is a large scale problem in which the number of

weapons and targets is large, making enumeration techniques impractical. For this reason, there is no exact algorithm to solve WA problem and good heuristics for solving this problem should be developed [13]. Also, since there exists no exact algorithms, it is not known how accurate the existing solutions obtained by these heuristic algorithms are [12].

The problem has two versions, namely *static* and *dynamic*. And, it can be classified as *target-based* and *asset-based*. Since this thesis mainly focuses on Weapon - Target Allocation problem, problem formulation and algorithms for the problem are given in the following chapters, in detail.

1.2 Scope and Objectives

Although the basic information on air defense, TEWA system, the process of Threat Evaluation and algorithms existing in the literature are described to provide a better understanding of the process, the main focus and scope of this study is on Weapon-Target Allocation problem.

The main objective of this study is to suggest an efficient algorithm for static target-based Weapon-Target Allocation problem that can be used in real ground-based air defense systems. For this reason, we first investigate and implement existing algorithms suggested for this special NP-Complete problem. Next, we define and relate the problem to some other known problems and use those algorithms to solve Weapon-Target Allocation problem.

Making a comparison of all algorithms for the problem in terms of correctness and efficiency is also one of our objectives.

Although, we studied the static target-based Weapon-Target Allocation Problem, the algorithms can be adopted to the dynamic or asset-based versions. However, in the dynamic version, there may be rapid changes in the assignments and some fixing rules may be needed. Our scope is the static target-based version of the problem.

1.3 Contributions of the Thesis

In this thesis, static Weapon-Target Allocation problem is defined and studied in detail. A literature review of existing studies about the topic is conducted. Various algorithms *(search algorithms, maximal marginal return algorithms, evolutionary algorithms, bipartite graph matching algorithms)* are applied to the problem and implemented. Adjusting the problem to be able to use the existing algorithms and improving the existing algorithms by making use of the maximum allowed time criterion is one of the contributions of this thesis. Another contribution is that a testbed is developed to be able to compare the algorithms and allow users to implement new algorithms and compare them. Also, an algorithm is proposed by making use of the algorithms that give better results. This algorithm is compared to the other algorithms and it is shown that the suggested algorithm gives the best (closest to the optimal solution) result among all considered algorithms. The study and comparison of the algorithms for the Weapon - Target Allocation problem in terms of optimality, performance and efficiency criteria is also one of the main contributions of this thesis.

1.4 Organization of the Thesis

This thesis work contains 5 chapters. Chapter 2 is about the weapon - target allocation problem. Related work about the weapon-target allocation problem is given in this chapter. Several algorithms for the problem are defined and described in detail in Chapter 3. The developed algorithms are tested on both small and large scale problem instances and algorithms are compared based on optimality and performance criteria in Chapter 4. This chapter also includes the description about the simulation environment and computational results of the experiments. Chapter 5 concludes the thesis and states future work.

CHAPTER 2

WEAPON - TARGET ALLOCATION PROBLEM

2.1 Problem Formulation

Weapon - Target Allocation problem is the problem of engaging weapons to targets for the purpose of minimizing the damage on the defended assets. There are two versions of the Weapon - Target Allocation problem: *static* and *dynamic*.

Static Version: In the static version, all of the inputs to the problem are fixed, and all weapons are engaged to targets in a single stage. Weapons are assigned and fired simultaneously. The damage assessment is made after all weapon - target engagements are completed [12].

Dynamic Version: In the dynamic version, weapons are assigned in stages with the assumption that the outcomes of the weapon-target engagements of the previous stage are observed before assignments for the present stage are made [13].

Although, the static version of the problem has been studied in the literature, the study on the dynamic version is scant [13]. It may be due to the fact that the static version is a special case of the dynamic version.

Lloyd and Witsenhausen has proved that the static version of the Weapon - Target Allocation problem is an NP-Complete problem by formulating the problem as a nonlinear integer programming problem [14]. Since the static version of the problem is a special case of the dynamic one, the dynamic Weapon - Target Allocation problem is also an NP-Complete problem.

Static Weapon - Target Allocation Problem Definition

The problem can be classified as *target-based* and *asset-based*. If the aim is to assign weapons to targets so that the expected survivability of the target is minimum, it is *target-based weapon-target allocation problem*. On the other hand, if the aim is to assign weapons to targets so that the expected survivability of the defended assets is maximum, then it is *asset-based weapon-target allocation problem*.

In the *asset-based* version of the problem, it is assumed that the aims of the targets are known. Since the *target-based* version is more appropriate when the aim of targets are not known for sure, this thesis work is focused on *target-based* weapon-target allocation problem. Yet, both *target-based* and *asset-based* problem formulations are given below.

Target-Based Weapon - Target Allocation Problem

$|T|$**:** The number of targets.

T**:** The set of targets. $T = \{T_1, T_2, ..., T_{|T|}\}$

$|W|$**:** The number of weapons.

W**:** The set of weapons. $W = \{W_1, W_2, ..., W_{|W|}\}$

V_i**:** The value of target i, calculated during the TE process. $i = 1, 2, ..., |T|$

P_{ik}**:** The probability that weapon k destroys target i if it is assigned to it (kill probability). $k = 1, 2, ..., |W|$ and $i = 1, 2, ..., |T|$

X_{ik}**:** 1 if weapon k is assigned to target i. 0, otherwise. $k = 1, 2, ..., |W|$ and $i = 1, 2, ..., |T|$

The problem can be formalized as the nonlinear integer programming problem. It is the problem of minimizing the total expected value of the surviving targets so that the total number of weapons used is no more than those available (Eqn. 2.1).

$$\min F = \sum_{i=1}^{|T|} V_i \prod_{k=1}^{|W|} (1 - P_{ik})^{X_{ik}} \tag{2.1}$$

subject to

$$\sum_{i=1}^{|T|} X_{ik} = 1 \tag{2.2}$$

$k = 1, 2, ..., |W|$ and $X_{ik} = 0, 1$.

In Eqn. 2.1, the product represents the expected probability of survival for target T_i. And minimizing the objective function value F, minimizes the expected total value of surviving targets.

A solution to this problem gives the allocation of weapons to targets so that the total objective function value F is minimum and the allocations can be represented as a matrix of decision variables:

$$\begin{pmatrix} X_{11} & X_{12} & ... & X_{1W} \\ X_{21} & X_{22} & ... & X_{2W} \\ ... & ... & X_{ik} & ... \\ X_{T1} & X_{T2} & ... & X_{TW} \end{pmatrix}$$

For a problem instance with $|T|$ targets and $|W|$ weapons, there are $|T|^{|W|}$ feasible solutions. So, even if there are 5 weapons and 5 targets, there are $5^5 = 3125$ possible allocations, and it is far from trivial for a decision maker to find the best of the solutions in real-time.

This formulation is the simplified version of static Weapon - Target Allocation problem. It is possible to add new constraints such as lower/upper bounds on the number of weapons of type i assigned to a target j, or lower/upper bounds on the number of weapons assigned to target j, or a lower bound on the survival value of the target j [12].

Asset-Based Weapon - Target Allocation Problem

$|T|$**:** The number of targets.

T**:** The set of targets. $T = \{T_1, T_2, ..., T_{|T|}\}$

$|W|$**:** The number of weapons.

W**:** The set of weapons. $W = \{W_1, W_2, ..., W_{|W|}\}$

$|A|$**:** The number of defended assets.

A**:** The set of defended assets. $A = \{A_1, A_2, ..., A_{|A|}\}$

ω_j**:** Protection value of defended asset A_j. $j = 1, 2, ..., |A|$

G_j**:** The set of targets aimed at the defended asset A_j. $j = 1, 2, ..., |A|$

Π_i**:** The probability that target i destroys the defended asset. $i = 1, 2, ..., |T|$

P_{ik}**:** The probability that weapon k destroys target i if it is assigned to it (kill probability). $k = 1, 2, ..., |W|$ and $i = 1, 2, ..., |T|$

X_{ik}**:** 1 if weapon k is assigned to target i. 0, otherwise. $k = 1, 2, ..., |W|$ and $i = 1, 2, ..., |T|$

The asset-based version of the problem is the problem of maximizing the total expected protection value of surviving defended assets. (Eqn. 2.3).

$$\max G = \sum_{j=1}^{|A|} \omega_j \prod_{i \epsilon G_j} (1 - \Pi_i \prod_{k=1}^{|W|} (1 - P_{ik})^{X_{ik}}) \tag{2.3}$$

subject to

$$\sum_{i=1}^{|T|} X_{ik} = 1 \tag{2.4}$$

$k = 1, 2, ..., |W|$ and $X_{ik} = 0, 1$.

In Eqn. 2.3, the inner product represents the expected probability of survival for target T_i as in the case of target-based version. In addition to that, the outer product represents the expected probability of survival for the defended asset A_j. And maximizing the objective function value G, maximizes the total expected protection value of the defended assets.

In addition to the allocation of weapons to targets, a solution to the asset-based version of the problem gives the assets that should be protected.

2.2 Related Work on Weapon - Target Allocation Problem

Since the algorithms are studied in detail in Chapter 3, this section only includes a brief overview of the algorithms that exist in the literature.

In [12], four lower-bounding schemes on the survival value of targets are proposed to give a lower bound on the solution of Weapon - Target Allocation problem. These schemes are described below:

1. **Using Generalized Integer Network Flow Formulation**

   The Weapon - Target Allocation problem is formulated as a generalized integer network flow problem on an appropriately defined network and a lower bound on the solution is derived with 2 different ways from the formulation.

   (a) **Linear Programming Based:** The lower bounds generated by this scheme do not seem to be very tight.

   (b) **Mixed Integer Programming Based:** The lower bounds generated by this scheme seem to be fairly tight.

2. **Minimum Cost Flow Based**

   The objective function is interpreted as maximizing the expected total damage to the targets. The problem is formulated as a maximum cost flow problem and an upper bound on the expected damage to the targets is found. Then, the found upper bound is subtracted from the total value of targets and a lower bound on the minimum survival value is found.

3. **Maximum Marginal Return Based**

   Maximum marginal return based scheme uses a greedy approach to obtain a lower bound. It is based on underestimation of the survival of a target when hit by a weapon since it is assumed that each target is hit by the best weapons.

Several exact and heuristic algorithms such as Branch and Bound, Maximum Marginal Return, Genetic Algorithms have been proposed to solve Weapon -

Target Allocation problem. Some of them are described briefly below:

1. **Branch and Bound Algorithms**

   In [12], four branch and bound algorithms are developed and compared for each lower bounding scheme given above. It is seen that, using the mixed integer programming based lower bounding scheme gives the most consistent results and is able to solve the highest size problems.

   In branch and bound algorithms, instead of enumerating all possible assignments, unnecessary enumeration steps are deleted and this improves efficiency [15]. Each node is a variable and the lower and upper bounds at that node is stored. Maximum marginal return strategy is used for branching. For each node of the tree, weapon-target combination giving the best improvement is found and the corresponding variable is set as the one to be branched on next.

   For searching, it is seen that for smaller size problems, breadth-first search strategy gives better results. However, for larger size problems, depth-first search strategy gives a better performance.

2. **Very Large Scale Neighborhood Search (VLSN) Algorithm**

   It is a neighborhood search algorithm where the size of the neighborhood is very large. It starts with a feasible solution and improves it by replacing it by an improved neighbor until it obtains a locally optimal solution [12].

3. **Maximum Marginal Return (MMR) Algorithm**

   It is a greedy algorithm in which the best weapon-target pair available is selected first. The change in the objective function from assigning each weapon-to-target pairing is evaluated then. And the weapon-to-target pairing that yields the best marginal return is selected as the next weapon-to-target pair. This process is iterated until all weapons are assigned to a target.

   Since it is a greedy algorithm, it finds the solution quickly and it is easy to implement [12].

If all weapons are identical, meaning that $W_i = W$ and $P_{ij} = P_j$, this algorithm gives the optimal solution [13]. However, if all weapons are not identical, it does not necessarily find the optimal solution. Usually, it is the case that it finds a solution which is far from the optimal solution.

4. **Genetic Algorithms (GA)**

   In [16], a genetic algorithm with greedy eugenics that takes into account a probability of kill value for each weapon is suggested and compared to MMR algorithm. It is seen that, although MMR algorithm runs much faster than GA, GA tends to find better solutions than MMR algorithm. And, GA efficiency increases as the number of targets and weapons increases. Also, it is seen that if a set of weapons can also hit a group of targets, meaning that grouping of weapons and targets is possible, this leads to faster and more optimal solutions.

CHAPTER 3

ALGORITHMS FOR WEAPON - TARGET ALLOCATION PROBLEM

3.1 Representations

Since over 15 algorithms are studied in this thesis, setting up a common representation for all of the algorithms will make the details easier to understand. By using Eqn. 2.1 given in Section 2, inputs and outputs are represented as below.

Input: ***(Problem)***

$|T|$**:** Number of targets. *(noOfTargets)*

$|W|$**:** Number of weapons. *(noOfWeapons)*

V: Target values. It is a $1x|T|$ matrix and the value of each element in the matrix is between 0 and 1, inclusive. *(targetValues)*

P: Kill probabilities. It is a $|T|x|W|$ matrix and the value of each element in the matrix is between 0 and 1, inclusive. *(killProbabilities)*

In this study, the number of targets and number of weapons are taken from the user. Then, target values and kill probabilities are generated randomly.

Output: ***(Solution)***

X: Weapon - target allocations. It is a $|T|x|W|$ matrix.

In this study, a solution is represented as a vector of length $|W|$ *(allocations)* for the simplicity. In the solution vector, each element takes a value between 1 and $|T|$. For example, if the value of 4^{th} element is 5 in the solution vector, it means that weapon 4 is allocated for target 5.

By using this kind of representation for the solution, a mapping from the decision variables (given as X in the problem formulation) is also provided.

F: Objective function value. It is calculated using Eqn. 2.1 given in Section 2 *(solutionValue)*. The pseudocode for the objective function value calculation algorithm is given as Algorithm 1.

Algorithm 1 Calculate Solution Value

```
solutionValue ← 0
i ← 1
while i <= noOfTargets do
  k ← 1
  while k <= noOfWeapons do
    if allocations[k] == i then
      targetValues[i] ← targetValues[i] * (1 − killProbabilities[i][k])
    end if
    k ← k + 1
  end while
  solutionValue ← solutionValue + targetValues[i]
  i ← i + 1
end while
return solutionValue
```

Since more than one weapon can be assigned to a single target, there should be no 0 in the solution vector. That is, all weapons should be allocated to a target even if there is only one target. Note that, if the number of targets is greater than number of weapons there may be unallocated targets left.

It is important to understand the input and output representations for the rest, so an example is given below:

Example:

Let the inputs are given as:

- $|T| = 3$
- $|W| = 5$
- V =
$$\begin{pmatrix} 0.8 & 0.95 & 0.6 \end{pmatrix}$$
- P =
$$\begin{pmatrix} 0.6 & 0.75 & 0.5 & 0.4 & 0.8 \\ 0.3 & 0.45 & 0.2 & 0.6 & 0.8 \\ 0.8 & 0.75 & 0.6 & 0.3 & 0.45 \end{pmatrix}$$

After the algorithm is applied, let the weapon - target allocation matrix (X) be

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

So, the corresponding solution vector is

$$\begin{pmatrix} 1 & 2 & 3 & 1 & 3 \end{pmatrix}$$

This solution is interpreted as 1^{st} weapon is assigned to 1^{st} target, 2^{nd} weapon is assigned to 2^{nd} target, 3^{rd} weapon is assigned to 3^{rd} target, 4^{th} weapon is assigned to 1^{st} target and 5^{th} weapon is assigned to 3^{rd} target.

The calculated objective function value for the given example is 0.8465.

In this study, basically four different types of algorithms are applied to the Weapon - Target Allocation problem, namely, *Search Algorithms*, *Maximum Marginal Return Algorithms*, *Evolutionary Algorithms*, and *Bipartite Graph Matching Algorithms*. At the end of this chapter, a hybrid algorithm that gives the best solution compared to the other algorithms is suggested.

In each section, first the general description of the basic algorithm is given. And for each basic algorithm, two or more algorithms that are derived from the

basic algorithm are described. When it will be more clear, the pseudocode of the algorithm is also provided. The evaluation and comparison of algorithms in terms of time and optimality is given in Section 4.

3.2 Search Algorithms

A search algorithm is an algorithm that finds an element with given propertied among a collection of elements. The Weapon - Target Allocation problem can be represented as a searching problem. But, in this problem, the element to be searched is dependent on all other elements constituting the collection. It is the element giving the minimum objective function value among all other elements.

3.2.1 Exhaustive Search Algorithm

An exhaustive search algorithm is an algorithm where the search is guaranteed to produce all reachable states before it terminates. It tries every possible way to search for a solution. Exhaustive search algorithms are also known as *brute-force* or *blind search algorithms* in the literature.

An exhaustive search algorithm is usually simple to implement, and always finds a solution if it exists. However, the cost of the algorithm is directly proportional to the number of possible solutions. For this reason, exhaustive search algorithms are typically used when the problem size is limited or when computation time is not as important as simplicity.

Here are some exhaustive search algorithms:

1. **Breadth-First Search Algorithm**

   In the breadth-first search, the search starts at root node and proceeds by generating and testing each node that is reachable from a parent node before it expands any of the children. To make it more clear, the visiting order of nodes is given in Figure 3.1.

2. **Depth-First Search Algorithm**

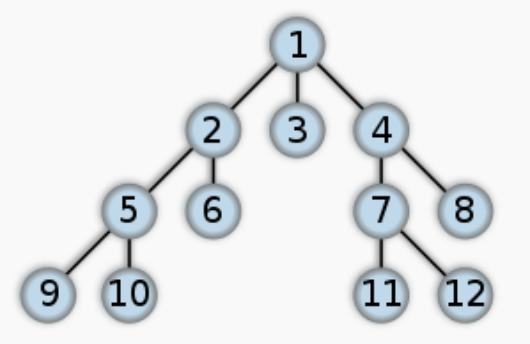

Figure 3.1: Visiting Orders of Nodes in Breadth-First Search

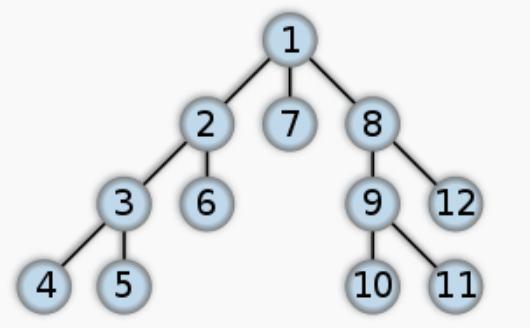

Figure 3.2: Visiting Orders of Nodes in Depth-First Search

In the depth-first search, the search starts at root node and proceeds as far as possible along each branch. Then, the search backtracks to the previous level and explores any remaining alternatives at this level, and so on. To make it more clear, the visiting order of nodes is given in Figure 3.2.

3. **Iterative Deepening Depth-First Search Algorithm**

   This algorithm is also known as *Depth-First Iterative Deepening*. In the iterative deepening depth-first search, the depth-limited search is run repeatedly. It increases the depth limit with each iteration until it reaches the depth of the shallowest goal state. Actually, iterative deepening depth-first search is equivalent to the breadth-first search, but it uses much less memory. On each iteration, it visits the nodes in the search tree in the same order as depth-first search, but the cumulative order in which nodes are first visited is effectively breadth-first.

To find a solution to the problem of Weapon - Target Allocation using Exhaustive Search, it is needed to search for every possible solutions, meaning that all nodes should be visited in the tree. For this reason, whether applying *Breadth-First Search* or *Depth-First Search* does not matter much.

For Weapon - Target Allocation problem, there are $|T|^{|W|}$ possible solutions to test. The pseudocode of the Exhaustive Algorithm is given as Algorithm 2.

Algorithm 2 Exhaustive Search Algorithm

1: $solution.allocations \leftarrow \{\}$
2: $solution.value \leftarrow MaxValue$
3: $noOfIterations \leftarrow |T|^{|W|}$
4: **if** $noOfIterations > K * allowedSearchTimeinMs$ **then**
5: **return** $NULL$
6: **end if**
7: $iterationNo \leftarrow 0$
8: **while** $iterationNo < noOfIterations$ **do**
9: $feasibleAllocations \leftarrow GetFeasibleAllocations(iterationNo)$
10: $feasibleSolutionValue \leftarrow CalculateSolutionValue(feasibleAllocations)$
11: **if** $feasibleSolutionValue < solution.value$ **then**
12: $solution.allocations \leftarrow feasibleAllocations$
13: $solution.value \leftarrow feasibleSolutionValue$
14: **end if**
15: $iterationNo \leftarrow iterationNo + 1$
16: **end while**
17: **return** *solution*

Exhaustive search algorithm first gets a feasible solution *(9)*, then calculates the solution value for that feasible solution *(10)*. If the feasible solution value is less than the previously found solution value *(11)*, it replaces the solution with this feasible solution *(12, 13)* since it is a better solution. This procedure is repeated until all feasible solutions are tested *(8 - 16)* and the solution is returned *(17)*.

The pseudocode of getting feasible allocations technique is given as Algorithm 3.

Algorithm 3 Get Feasible Allocations

$feasibleAllocations \leftarrow \{\}$
while $iterationNo >= 0$ **do**
  $feasibleAllocations.Add(iterationNo \ \% \ noOfTargets + 1)$
  $iterationNo \leftarrow iterationNo/noOfTargets$
end while
while $feasibleAllocations.Count < noOfWeapons$ **do**
  $feasibleAllocations.Add(1)$
end while
return $feasibleAllocations$

For example if $|T| = 3$ and $|W| = 2$, there are $3^2 = 9$ feasible solutions and these feasible solutions are found as {1, 1}, {2, 1}, {3, 1}, {1, 2}, {2, 2}, {3, 2}, {1, 3}, {2, 3}, {3, 3} using that technique. Note that {1, 1} means that both weapons are assigned to the first target and {3, 2} means that the first weapon is assigned to the third target and the second weapon is assigned to the second target, and so on.

K is a constant. In the experiments of exhaustive search algorithm, it is calculated that when the number of iterations is 46656, the computation time of the algorithm is nearly 125 ms on the computer whose specifications are given in *Section 4.2*. So, the value of K is calculated as $46656/125 = 370$ for that machine. For this reason, if the number of iterations is greater than 370 times the maximum allowed search time, then the algorithm should not be applied *(5)* because it cannot return within the time of maximum allowed search.

Exhaustive search algorithm is easy to implement and since it tests all of the feasible solutions, it finds the optimal solution. So, if the number of iterations is small enough, then it is useful to apply exhaustive search, since it finds the best solution.

3.2.2 Random Search Algorithm

A random search algorithm is an algorithm that uses some kind of randomness or probability [18]. Random search algorithms are nondeterministic algorithms meaning that they may give different solutions at each run. Particle swarm optimization, ant colony optimization and genetic algorithms are some examples of random search algorithms. However, these algorithms are given in the section of *Evolutionary Algorithms*. In this section, only a naive random search algorithm is given (Algorithm 4).

Algorithm 4 Random Search Algorithm

```
1:  startTime ← Now
2:  endTime ← startTime + allowedSearchTime
3:  solution.allocations ← {}
4:  solution.value ← MaxValue
5:  while endTime < Now do
6:      weaponNo ← 1
7:      while weaponNo <= noOfWeapons do
8:          targetNo ← GetARandomTarget
9:          feasibleAllocations.Add(targetNo)
10:         weaponNo ← weaponNo + 1
11:     end while
12:     feasibleSolutionValue = CalculateSolutionValue(feasibleAllocations)
13:     if feasibleSolutionValue < solution.Value then
14:         solution.allocations = feasibleAllocations
15:         solution.value = feasibleSolutionValue
16:     end if
17: end while
18: return solution
```

Random search algorithm first creates a feasible solution by assigning a target randomly for each weapon *(7 - 14)*. Then, it calculates the solution value for that feasible solution *(12)*. If the feasible solution value is less than the previously found solution value, it replaces the solution value with this feasible solution

since it is a better solution *(13 - 15)*. This procedure is repeated until the maximum allowed search time expires *(5 - 17)*. And the solution is returned *(18)*.

As exhaustive search algorithm, random search algorithm is also easy to implement and it finds solution very efficiently. However, the solution found by applying random search algorithm is not the optimal or best solution. Even, there is a chance that the algorithm returns the worst solution (if time is not enough, it returns the first created feasible solution without testing the other ones). Yet, a common experience is that random search algorithms perform well and are robust in the sense that they give useful information quickly provided that they explore the solution space adequately [18].

3.3 Maximum Marginal Return (MMR) Algorithms

Maximum marginal return algorithms are algorithms that assign weapons sequentially with each weapon being assigned to the target which results in the maximum decrease (marginal return) in the objective function value [19]. In other words, in maximum marginal return algorithms, a weapon is always assigned to the target with maximum improvement in the objective function value.

Maximum marginal return algorithms are heuristic algorithms, they are easy to implement and efficient algorithms. Although these algorithms don't give the optimal or best solution it is known that these algorithms give near optimal solutions.

Greedy MMR, Random MMR and Advanced MMR algorithms are described in the following subsections.

3.3.1 Greedy MMR Algorithm

Greedy MMR algorithm is a deterministic algorithm that finds a target for each weapon so that the maximum decrease on the solution value occurs. A greedy MMR algorithm is given as Algorithm 5.

Algorithm 5 Greedy MMR Algorithm

```
1: solution.allocations ← {}
2: solution.value ← MaxValue
3: k ← 1
4: while k <= noOfWeapons do
5:    maxDecrease ← MinValue
6:    i ← 1
7:    while i <= noOfTargets do
8:       decrease ← targetValues[i] * killProbabilities[i][k]
9:       if decrease > maxDecrease then
10:         maxDecrease ← decrease
11:         allocatedTarget ← i
12:      end if
13:      i ← i + 1
14:   end while
15:   solution.allocations[k] ← allocatedTarget
16:   targetValues[allocatedTarget] ← targetValues[allocatedTarget] −
      maxDecrease
17:   k ← k + 1
18: end while
19: solution.value = CalculateSolutionValue(solution.allocations)
20: return solution
```

Greedy MMR algorithm sequentially assigns the target so that the maximum decrease in the solution value occurs for each weapon *(4 - 18)*. Then, it calculates the solution value *(19)* and returns the solution *(20)*.

3.3.2 Greedy MMR Algorithm Improved with Local Search

Greedy MMR algorithm given in the previous section is independent of the maximum allowed search time. It finds the solution in one iteration. It is very quick and deterministic. However, it can be improved by making use of the maximum allowed search time. The pseudocode is given in Algorithm 6.

Algorithm 6 Greedy MMR Algorithm Improved with Local Search

```
startTime ← Now
endTime ← startTime + allowedSearchTime
solution ← ApplyGreedyMmrAlgorithm()
while endTime < Now do
    neighborAllocations ← RandomlySwapAllocations(solution.allocations)
    neighborSolutionValue ← CalculateSolutionValue(neighborAllocations)
    if neighborSolutionValue < solution.solutionValue then
        solution.allocations ← neighborAllocations
        solution.solutionValue ← neighborSolutionValue
    end if
end while
return solution
```

Note that, although Greedy MMR algorithm is deterministic, this improved version is nondeterministic since it uses randomization in the local search process.

3.3.3 Random MMR Algorithm

Random MMR algorithm is a nondeterministic algorithm that randomly selects a weapon and finds a target for that weapon so that the maximum decrease on the solution value occurs. The random MMR algorithm is given as Algorithm 7.

Random MMR algorithm first selects a weapon from unallocated weapons list

Algorithm 7 Random MMR Algorithm

1: $startTime \leftarrow Now$
2: $endTime \leftarrow startTime + allowedSearchTime$
3: $solution.allocations \leftarrow \{\}$
4: $solution.value \leftarrow MaxValue$
5: **while** $endTime < Now$ **do**
6: $allocatedWeaponCount \leftarrow 0$
7: **while** $allocatedWeaponCount < noOfWeapons$ **do**
8: $k \leftarrow GetARandomWeapon(unallocatedWeapons)$
9: $maxDecrease \leftarrow MinValue$
10: $i \leftarrow 1$
11: **while** $i < noOfTargets$ **do**
12: $decrease \leftarrow targetValues[i] * killProbabilities[i][k]$
13: **if** $decrease > maxDecrease$ **then**
14: $maxDecrease \leftarrow decrease$
15: $allocatedTarget = i$
16: **end if**
17: $i \leftarrow i + 1$
18: **end while**
19: $allocations[k] \leftarrow allocatedTarget$
20: $targetValues[allocatedTarget] \leftarrow targetValues[allocatedTarget] - maxDecrease$
21: $unallocatedWeapons.Remove(k)$
22: $allocatedWeaponCount \leftarrow allocatedWeaponCount + 1$
23: **end while**
24: $solutionValue \leftarrow CalculateSolutionValue(allocations)$
25: **if** $solutionValue < solution.Value$ **then**
26: $solution.allocations \leftarrow allocations$
27: $solution.Value \leftarrow solutionValue$
28: **end if**
29: **end while**
30: **return** *solution*

randomly *(8)*. Then, it finds the target so that the maximum decrease in the solution value occurs for that weapon *(11 - 18)*. It repeats this procedure until a feasible solution (no weapons should be left unassigned) is get *(8 - 23)*. Next, it calculates the solution value for that feasible solution *(24)*. If the feasible solution value is less than the previously found solution value, it replaces the solution with this feasible solution since it is a better solution *(25 - 28)*. This procedure is repeated until the maximum allowed search time is expired *(5 - 29)*. And the solution is returned *(30)*.

3.3.4 Advanced MMR Algorithm

Advanced MMR algorithm finds weapon-target pairs so that the maximum decrease on the solution value occurs. The choice of which weapon is allocated next is based on which weapon-target pair maximizes the marginal return. This version of the algorithm is given as Algorithm 8.

Advanced MMR algorithm first selects a weapon-target pair so that the maximum decrease in the solution value occurs *(7 - 19)*. It repeats this procedure until no weapon is left unassigned *(4 - 24)*. Then, it calculates the solution value *(25)* and returns the solution *(26)*.

3.3.5 Advanced MMR Algorithm Improved with Local Search

Advanced MMR algorithm given in the previous section is independent of the maximum allowed search time. It finds the solution in one iteration. It is very quick and deterministic. However, it can be improved by making use of the maximum allowed search time. The pseudocode is given in Algorithm 9.

Note that, although Advanced MMR algorithm is deterministic, this improved version is nondeterministic since it uses randomization in the local search process.

Algorithm 8 Advanced MMR Algorithm

```
1: solution.allocations ← {}
2: solution.value ← MaxValue
3: allocatedWeaponCount ← 0
4: while allocatedWeaponCount < noOfWeapons do
5:    maxDecrease ← MinValue
6:    k ← 1
7:    while k < unallocatedWeapons.Count do
8:       i ← 1
9:       while i < noOfTargets do
10:         decrease ← targetValues[i] * killProbabilities[i][k]
11:         if decrease > maxDecrease then
12:            maxDecrease ← decrease
13:            allocatedTarget ← i
14:            allocatedWeapon ← k
15:         end if
16:         i ← i + 1
17:      end while
18:      k ← k + 1
19:   end while
20:   unallocatedWeapons.Remove(allocatedWeapon)
21:   solution.allocations[k] ← allocatedTarget
22:   targetValues[allocatedTarget] ← targetValues[allocatedTarget] −
      maxDecrease
23:   allocatedWeaponCount ← allocatedWeaponCount + 1
24: end while
25: solution.value ← CalculateSolutionValue(solution.allocations)
26: return solution
```

Algorithm 9 Advanced MMR Algorithm Improved with Local Search

```
startTime ← Now
endTime ← startTime + allowedSearchTime
solution ← ApplyAdvancedMmrAlgorithm()
while endTime < Now do
    neighborAllocations ← RandomlySwapAllocations(solution.allocations)
    neighborSolutionValue ← CalculateSolutionValue(neighborAllocations)
    if neighborSolutionValue < solution.solutionValue then
        solution.allocations ← neighborAllocations
        solution.solutionValue ← neighborSolutionValue
    end if
end while
return solution
```

3.4 Evolutionary Algorithms

Evolutionary algorithms are algorithms that mimic the natural biological evolution. In evolutionary algorithms, individuals constitute the population and each population represents a candidate solution to the optimization problem. Survival of the fittest principle is applied to produce better approximation to the solution and fitness function provides the evolution of populations of individuals that are better suited to the environment. The process of selecting individuals as parents according to their level of fitness and cross-overing the parents to create a new generation is applied. And mutation operator is applied to the newly created generation. The purpose of this process is to approach closer to the solution.

Evolutionary algorithms model natural evolution processes, such as selection, recombination, mutation, migration, locality and neighborhood. In Figure 3.3, a flow chart of a simple evolutionary algorithm is given.

An evolutionary algorithm first generates an initial population randomly. This created population is also a feasible solution. Next, it calculates the solution value for that population or feasible solution. If the optimization criteria are

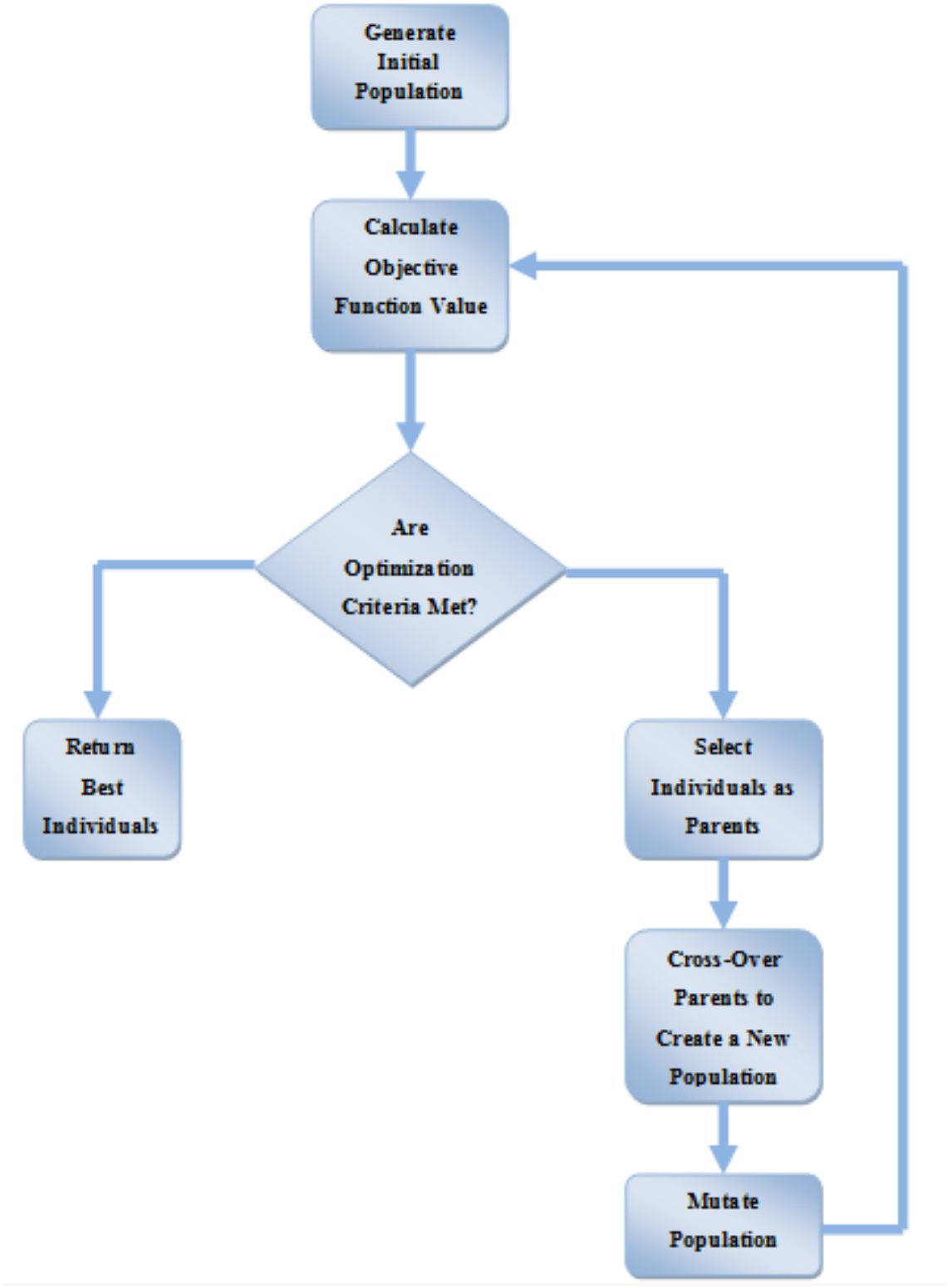

Figure 3.3: Flow Chart of Evolutionary Algorithms

not met, a new generation is created. In the generation creation process, firstly, individuals are selected as parents according to their fitness (selection). And parents are recombined to create generation (cross-over). Finally, the mutation operator is applied to the newly generated creation and the solution value is calculated for this generation or feasible solution. This process is repeated until the optimization criteria is met and the algorithm stops and returns the lastly generated population.

Operators Applied in Evolutionary Algorithms: [20]

1. **Selection:** The selection operator is applied to determine which individuals are chosen for cross-over or recombination and how many individuals each selected individual produces. First a fitness assignment is made for each individual and then parents are selected according to their fitness. In the selection process, there are many techniques that can be applied such as roulette-wheel selection, stochastic universal sampling or tournament selection.

   Roulette-Wheel Selection: It is the simplest selection technique. It is also known as stochastic sampling with replacement. In this technique, the individuals are selected based on a fitness-proportional procedure. The individuals are mapped to a line such that each individual's segment is equal in size to its fitness. Then a random number is generated and the individual whose segment spans the random number is selected. This process is repeated until the desired number of individuals is obtained.

   To better understand, here is an example: Assume that there are 5 individuals and the fitness values of the individuals are 1, 0.2, 1.6, 0.4 and 1.8. So, the selection probabilities of the individuals are $1/5 = 0.2$, $0.2/5 = 0.04$, $1.6/5 = 0.32$, $0.4/5 = 0.08$, $1.8/5 = 0.36$. And assume that two individuals will be selected and random numbers that are generated are 0.5 and 0.89. Then, according to the Figure 3.3, 3^{rd} and 2^{nd} individuals will be selected.

   Stochastic Universal Sampling: As in the roulette-wheel selection,

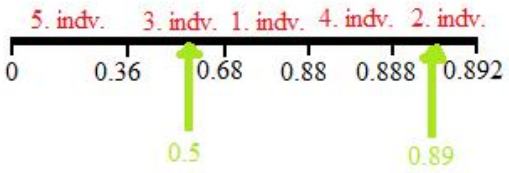

Figure 3.4: Roulette-Wheel Selection

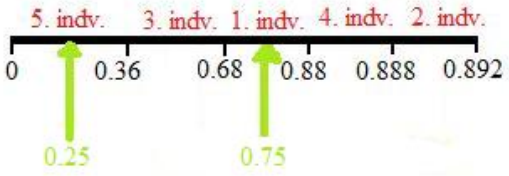

Figure 3.5: Stochastic Universal Sampling

the individuals are mapped to a line such that each individual's segment is equal in size to its fitness. However, in stochastic universal sampling technique, equally spaced pointers are placed over the line as many as there are individuals to be selected. The position of the first pointer is given by a randomly generated number.

To better understand, assume the example given in Roulette-Wheel Selection discussion. And assume that the randomly generated number is 0.25 for the position of the first pointer. The distance between pointers is $1/2 = 0.5$. Then, the position of the second pointer is calculated as $0.25 + 0.5 = 0.75$. So, according to the Figure 3.5, 5. and 1. individuals will be selected.

Tournament Selection: A number of individuals are chosen randomly from the population. The best individual from this group is selected as parent. This process is repeated as many times as the number of individuals that must be chosen.

2. **Cross-Over (Recombination):** The cross-over operator is applied to achieve genetic recombination. By this operator, two or more parents are combined to produce new individuals.

   Single-Point Cross-Over: In single-point cross-over, one position is se-

lected randomly and by exchanging the variables between the individuals about this position, two new individuals are generated.

To better understand, here is an example: Assume that each parent has 8 binary variables. Let the variables of the first parent be 01110101, the variables of the second parent be 10011010. And let the randomly selected cross-over position be 3. Then, after the single-point cross-over operator is applied, the generated individuals be 01111010 and 10010101.

Multi-Point Cross-Over: In multi-point cross-over, multiple positions are selected randomly with no duplicates and by exchanging the variables between the individuals about these positions, two new individuals are generated.

To better understand, assume the example given in single-point cross-over discussion. And let the randomly selected cross-over positions be 2, 4 and 7. Then, after the multi-point cross-over operator is applied, the generated individuals be 01010100 and 10111011.

3. **Mutation:** The mutation operator is applied to randomly alter the individuals. It is applied to the variables of the individuals that are generated by cross-over operator. When applying mutation operator, first a position is randomly selected and the value of the variable at that position is flipped for a binary variable.

   To better understand, here is an example: Let the variables of the individual to be mutated be 01010100. And the randomly generated mutation position be 5. Then, the individual created after applying the mutation operator be 01011100.

4. **Reinsertion:** The reinsertion operator is applied when the number of individuals that are produced by using cross-over and mutation operators is less than the size of the original population. Similarly, when the number of individuals that are produced is more than the size of the original population, reinsertion operator is applied to determine which individuals are to exist in the new population.

Genetic algorithms and swarm intelligence algorithms are the most popular types

of evolutionary algorithms. Swarm intelligence algorithms are algorithms that take inspiration from the social behaviors of insects and other animals [21].

Here are some examples of swarm-based algorithms:

1. **Ant-Colony Optimization:** It simulates the foraging behavior of ant colonies.
2. **Particle Swarm Optimization:** It simulates the movement or intelligence of swarms.
3. **Artificial Bee Colony:** It simulates the foraging behavior of honey bees.
4. **Firefly:** It simulates the flashing behavior of fireflies.
5. **Cuckoo Search:** It simulates the brooding behavior of cuckoo species.

In the following subsections, *genetic algorithm*, *ant-colony optimization* and *particle swarm optimization* algorithms are described in detail.

3.4.1 Genetic Algorithm

In this section, a sample genetic algorithm that is applied on Weapon - Target Allocation problem is described. Genetic algorithm is a kind of evolutionary algorithm that uses random search. Since the algorithm uses randomization, it is a nondeterministic algorithm. The pseudocode of the applied algorithm is given as Algorithm 10.

Genetic algorithm first generates an initial population *(10)* by using Algorithm 11.

Initial population is constructed from individuals and each individual is a feasible solution where each weapon is allocated to a random target. Then, for each individual or feasible solution in the population, it calculates the solution value and sets the solution to the individual that gives the minimum solution value

Algorithm 10 Genetic Algorithm

1: $startTime \leftarrow Now$
2: $endTime \leftarrow startTime + allowedSearchTime$
3: $solution.allocations \leftarrow \{\}$
4: $solution.value \leftarrow MaxValue$
5: **if** $noOfTargets > noOfWeapons$ **then**
6: $\quad noOfIndividuals \leftarrow noOfTargets$
7: **else**
8: $\quad noOfIndividuals \leftarrow noOfWeapons$
9: **end if**
10: $population \leftarrow GenerateInitialPopulation(noOfIndividuals)$
11: **while** $endTime < Now$ **do**
12: $\quad individualNo \leftarrow 1$
13: $\quad$ **while** $individualNo <= noOfIndividuals$ **do**
14: $\quad\quad solFromIndv \leftarrow population[individualNo]$
15: $\quad\quad solValueFromIndv \leftarrow CalculateSolutionValue(solFromIndv)$
16: $\quad\quad$ **if** $solValueFromIndv < solution.value$ **then**
17: $\quad\quad\quad solution \leftarrow solFromIndv$
18: $\quad\quad$ **end if**
19: $\quad\quad individualNo \leftarrow individualNo + 1$
20: $\quad$ **end while**
21: $\quad parents \leftarrow SelectParents(population)$
22: $\quad population \leftarrow CrossOver(parents)$
23: $\quad population \leftarrow Mutate(population)$
24: **end while**
25: **return** $solution$

Algorithm 11 Generate Initial Population

```
population ← {}
i ← 1
while i <= noOfIndividuals do
  individual ← {}
  k ← 1
  while k <= noOfWeapons do
    individual.Add(RandomlySelectATarget)
    k ← k + 1
  end while
  population.Add(individual)
  i ← i + 1
end while
return population
```

(13 - 20). Next, it applies selection operator to the population *(21)* by using Algorithm 12.

Deterministic Tournament Selection is applied as the selection operator technique. In the selection phase, each individual to be selected as parent is determined as follows: First, two individuals are selected from the population randomly. Then, their solution values are compared and the individual having less solution value is added to the list of parents. This process is repeated until all parents are selected. The genetic algorithm then applies multi-point cross-over operator to the parents that are selected from the selection phase *(22)* by using Algorithm 13.

In this phase, random positions are selected as the cross-over positions and the individuals that are created by exchanging parents from those cross-over positions are returned. Next, the algorithm applies mutation operator to the newly generated population *(23)* by using Algorithm 14.

In the mutation phase, a position is chosen randomly, and for the weapon at that position, a new target is allocated randomly. The individuals created by applying selection, cross-over and mutation operators constitute the new population. The

Algorithm 12 Select Parents

```
parents ← {}
i ← 1
while i <= noOfIndividuals do
  candidate1 ← population[RandomPosition]
  candidate2 ← population[RandomPosition]
  solutionValue1 ← candidate1.solutionValue
  solutionValue2 ← candidate2.solutionValue
  if solutionValue1 < solutionValue2 then
    parents.Add(candidate1)
  else
    parents.Add(candidate2)
  end if
  i ← i + 1
end while
return parents
```

whole process (calculating solution values for each individual in the population and generating new population) is repeated until the maximum allowed search time expires *(11 - 24)*. And the solution is returned as the individual that has the minimum solution value*(25)*.

3.4.2 Genetic Algorithm Improved with MMR Algorithms

Instead of generating the initial population randomly, using an MMR algorithm in the generation phase improves the solution. However, since applying an algorithm increases the generation time of the population, algorithm should be chosen carefully. In this work, *Greedy MMR* and *Advanced MMR* algorithms are tried and it is seen that they give good results. Since the comparison is made in Chapter 4, only the pseudocode and description are given here.

The pseudocodes of the methods that use Greedy MMR algorithm and Advanced MMR algorithm are given in Algorithm 15 and Algorithm 16, respectively.

Algorithm 13 Cross-Over Parents

$individuals \leftarrow parents$
$i \leftarrow 1$
while $i <= noOfIndividuals$ **do**
  $crossOverPosition \leftarrow RandomlySelectAPosition$
  $parent1 \leftarrow parents[i]$
  $parent2 \leftarrow parents[i+1]$
  $tmp \leftarrow parent2$
  while $crossOverPosition <= noOfWeapons$ **do**
    $parent2[crossOverPosition+1] \leftarrow parent1[crossOverPosition+1]$
    $parent1[crossOverPosition+1] \leftarrow tmp[crossOverPosition+1]$
    $crossOverPosition \leftarrow crossOverPosition+1$
  end while
  $individuals[i] \leftarrow parent1$
  $individuals[i+1] \leftarrow parent2$
  $i \leftarrow i+2$
end while
return *individuals*

Algorithm 14 Mutate Individuals

$i \leftarrow 1$
while $i <= noOfIndividuals$ **do**
  $mutationPosition \leftarrow RandomlySelectAPosition$
  $tmp \leftarrow individuals[i]$
  $tmp[mutationPosition] \leftarrow RandomlySelectATarget$
  $individuals[i] \leftarrow tmp$
  $i \leftarrow i+1$
end while
return *individuals*

Algorithm 15 Generate Initial Population using Greedy MMR

$population \leftarrow \{\}$

$mmrSol \leftarrow ApplyGreedyMmrAlgorithm()$

$population.Add(mmrSol)$

$i \leftarrow 1$

while $i < noOfIndividuals$ **do**

$neighbor.allocations \leftarrow RandomlySwapAllocations(mmrSol.allocations)$

$neighbor.solutionValue \leftarrow CalculateSolutionValue(neighbor.allocations)$

$population.Add(neighbor)$

$i \leftarrow i + 1$

end while

return *population*

Algorithm 16 Generate Initial Population using Advanced MMR

$population \leftarrow \{\}$

$mmrSol \leftarrow ApplyAdvancedMmrAlgorithm()$

$population.Add(mmrSol)$

$i \leftarrow 1$

while $i < noOfIndividuals$ **do**

$neighbor.allocations \leftarrow RandomlySwapAllocations(mmrSol.allocations)$

$neighbor.solutionValue \leftarrow CalculateSolutionValue(neighbor.allocations)$

$population.Add(neighbor)$

$i \leftarrow i + 1$

end while

return *population*

In the generation of the initial population, first individual is chosen to be the solution found by applying the Greedy/Advanced MMR. And the other individuals are chosen by swapping allocations of that solution. By doing so instead of generating the initial population randomly, better individuals are created and the solution is closer to the optimal solution.

3.4.3 Ant-Colony Optimization Algorithm

Ant-colony optimization takes inspiration from the foraging behavior of ant colonies.

Initially all of the ants search for the food randomly. When an ant finds a food, it starts to deposit pheromone on the ground while returning back to the colony. By depositing pheromone on the ground, they mark the path to the food that should be followed by other members of the colony. If an ant comes across a path with pheromone, it stops searching for the food randomly and starts to follow the path marked with pheromone. If it reaches the food, it starts to deposit pheromone on the path back to the colony also. This positive feedback strengthens the pheromone trail on the same path and causes all of the ants to follow a single path. On the other hand, if the path is not followed by other colony members, the pheromone evaporates in time and eventually the path disappears.

An Ant-Colony Optimization algorithm basically consists of 3 main steps [21]. After the initialization of pheromone trails, while there is still time, at each iteration:

1. Ants create solutions.
2. Created solutions are improved through a local search. This process is also known as daemon actions and it is an optional process.
3. Pheromone update is applied to increase the pheromone values that are associated with good solutions and to decrease the pheromone values that are associated with bad solutions (pheromone evaporation).

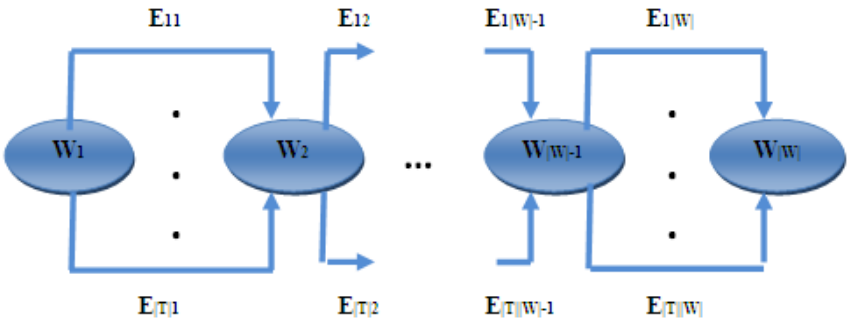

Figure 3.6: Graph Representation of Weapon - Target Allocation Problem

To be able to apply ant-colony optimization algorithm to a problem, the problem needs to be represented as a graph. The graph representation of Weapon - Target Allocation problem is shown in Figure 3.6.

Ant System is the first Ant-Colony Optimization algorithm proposed in the literature. *Max-Min Ant System* and *Ant Colony System* are two variants of the Ant System. In this study, *Ant Colony System* is chosen to be applied as an Ant-Colony Optimization algorithm [22].

The pseudocode of the Ant-Colony Optimization algorithm is given as Algorithm 17.

The description of the algorithm is given below in subsections of *Initialization*, *Construction of the Solution*, *Local Pheromone Update*, and *Global Pheromone Update*.

Initialization

In Ant Colony System, during the initialization phase, heuristic values *(10)* and pheromone values *(11)* are calculated.

For the Weapon - Target Allocation problem, the heuristic value (η_{ik}) for the edge E_{ik} is assigned as follows:

$$\eta_{ik} = V_i \times P_{ik} \tag{3.1}$$

And the value of the pheromone (τ_{ik}) for all the edges E_{ik} are assigned uniformly.

Algorithm 17 Ant-Colony Optimization Algorithm

```
1: startTime ← Now
2: endTime ← startTime + allowedSearchTime
3: solution.allocations ← {}
4: solution.value ← MaxValue
5: if noOfTargets > noOfWeapons then
6:    noOfAnts ← noOfTargets
7: else
8:    noOfAnts ← noOfWeapons
9: end if
10: CalculateHeuristicValues()
11: CalculatePheromoneValues()
12: while endTime < Now do
13:    minSolutionValue ← MaxValue
14:    antNo ← 1
15:    while antNo <= noOfAnts do
16:       constructedSol ← ConstructSolution()
17:       if constructedSol.solutionValue < minSolutionValue then
18:          bestSolValue ← constructedSol.solutionValue
19:          iterationBestSolAlloc ← constructedSol.allocations
20:          if constructedSol.solutionValue < solution.solutionValue then
21:             solution ← constructedSol
22:          end if
23:       end if
24:       CalculateHeuristicValues()
25:       antNo ← antNo + 1
26:    end while
27:    UpdatePheromoneValues(iterationBestSolAlloc, bestSolValue)
28: end while
29: return solution
```

The pseudocode of the calculation method of heuristic values is given as Algorithm 18.

Algorithm 18 Calculate Heuristic Values

$i \leftarrow 1$
while $i <= noOfTargets$ **do**
  $k \leftarrow 1$
  while $k <= noOfWeapons$ **do**
    $heuristicValues[i][k] \leftarrow targetValues[i] \times killProbabilities[i][k]$
    $k \leftarrow k + 1$
  end while
  $i \leftarrow i + 1$
end while

The pseudocode of the calculation method of pheromone values is given as Algorithm 19.

Algorithm 19 Calculate Pheromone Values

$i \leftarrow 1$
while $i <= noOfTargets$ **do**
  $k \leftarrow 1$
  while $k <= noOfWeapons$ **do**
    $pheromoneValues[i][k] \leftarrow 1/noOfAnts \times solutionValue$
    $k \leftarrow k + 1$
  end while
  $i \leftarrow i + 1$
end while

Construction of the Solution

During the construction of the solution phase *(16)*, the pseudorandom proportional rule given in Eqn. 3.2 is used.

For each weapon k, target i is chosen with the rule given in Eqn. 3.2 for Weapon - Target Allocation problem.

$$i = \begin{cases} argmax_{i\epsilon\{1,2,\ldots,|T|\}}\{\tau_{ik}^{\alpha}\eta_{ik}^{\beta}\}, & \mathrm{q} \leq q_0 \\ s, & \text{otherwise} \end{cases} \tag{3.2}$$

q is a random variable that is uniformly distributed over $[0, 1]$.

q_0 is a threshold and it is set to 0.5 for simplicity.

The parameters α and β are constants that control the relative importance of the pheromone versus the heuristic information. They are both set to 1 for simplicity.

s is an index selected using Roulette Wheel Selection. The probability that s is selected is given in Eqn. 3.3.

$$P_s = \frac{\tau_{sk}^{\alpha}\eta_{sk}^{\beta}}{\sum_{l\epsilon\{1,2,\ldots,|T|\}}\tau_{lk}^{\alpha}\eta_{lk}^{\beta}} \tag{3.3}$$

So, the interpretation of Eqn. 3.2 is as follows: If $q \leq q_0$, then the target maximizing the product of the pheromone and heuristic information is chosen for the weapon. Otherwise, the target which is determined through Roulette Wheel Selection is chosen for the weapon.

The pseudocode of the constructing the solution method is given as Algorithm 20.

The method for finding the target index for a weapon based on Eqn. 3.2 is given as Algorithm 21.

Local Pheromone Update

After each construction phase (*i.e,* an ant have reached node $W_{|W|}$), the local pheromone update is applied by using Eqn. 3.4.

$$\tau_{ik} = (1 - \varphi)\tau_{ik} + \varphi\tau_0 \tag{3.4}$$

φ is a constant to represent pheromone evaporation rate and it is set to 0.1 for

Algorithm 20 Construct Solution

$solution.allocations \leftarrow \{\}$

$solution.value \leftarrow MaxValue$

$k \leftarrow 1$

while $k <= noOfWeapons$ **do**

$i \leftarrow FindTargetIndexForWeapon(k)$

$solution.allocations.Add(i)$

$UpdatePheromoneValuesLocally(k)$

$targetValues[i] \leftarrow targetValues[i] \times (1 - killProbabilities[i][k]$

$CalculateHeuristicValues()$

$k \leftarrow k + 1$

end while

$solution.solutionValue \leftarrow CalculateSolutionValue(solution.allocations)$

return *solution*

simplicity.

τ_0 is the initial value of the pheromone.

The pseudocode of the algorithm that updates local pheromone values using Eqn. 3.4 is given as Algorithm 23.

Global Pheromone Update

At the end of each iteration (*i.e.,* all ants within an iteration have reached node $W_{|W|}$) the global pheromone update is applied by using Eqn. 3.5.

$$\tau_{ik} = \begin{cases} (1-\rho)\tau_{ik} + \rho\Delta\tau_{ik}, & \text{if weapon } k \text{ is allocated for target } i \\ \tau_{ik}, & \text{otherwise} \end{cases} \tag{3.5}$$

ρ is a constant to represent pheromone increase rate and it is set to the value of pheromone evaporation rate, 0.1 for simplicity.

$\Delta\tau_{ik}$ is the amount of pheromone laid on edge E_{ik} and it is computed using Eqn. 3.6.

Algorithm 21 Find Target Index For Weapon

$targetIndex \leftarrow 1$
$Q \leftarrow RandomValue$
if $Q <= Q_0$ **then**
  $targetIndex \leftarrow ArgMax(k)$
else
  $total \leftarrow 0$
  $i \leftarrow 1$
  while $i <= noOfTargets$ **do**
    $total \leftarrow total + pheromoneValues[i][k]^{\alpha} \times heuristicValues[i][k]^{\beta}$
    $i \leftarrow i + 1$
  end while
  $Q \leftarrow RandomValue * total$
  $total \leftarrow 0$
  while $i <= noOfTargets$ **do**
    $total \leftarrow total + pheromoneValues[i][k]^{\alpha} \times heuristicValues[i][k]^{\beta}$
    if $Q <= total$ **then**
      $targetIndex \leftarrow i$
      $break$
    end if
    $i \leftarrow i + 1$
  end while
end if
return $targetIndex$

Algorithm 22 Arg Max

1: $targetIndex \leftarrow 1$
2: $maxValue \leftarrow MinValue$
3: $i \leftarrow 1$
4: **while** $i <= noOfTargets$ **do**
5: $value \leftarrow pheromoneValues[i][k]^{\alpha} \times heuristicValues[i][k]^{\beta}$
6: **if** $value > maxValue$ **then**
7: $targetIndex \leftarrow i$
8: $maxValue \leftarrow value$
9: **end if**
10: $i \leftarrow i + 1$
11: **end while**
12: **return** $targetIndex$

Algorithm 23 Local Update Pheromone Values

1: $i \leftarrow 1$
2: **while** $i <= noOfTargets$ **do**
3: $pheromoneValues[i][k] \leftarrow pheromoneValues[i][k] \times evaporationRate$
4: $pheromoneValues[i][k] \leftarrow pheromoneValues[i][k] + (evaporationRate \times (1/noOfAnts \times solutionValue)$
5: $i \leftarrow i + 1$
6: **end while**
7: **return** $targetIndex$

$$\Delta\tau_{ik} = 1/F_{best} \tag{3.6}$$

The pseudocode of the algorithm that updates local pheromone values using Eqn. 3.5 is given as Algorithm 24.

Algorithm 24 Global Update Pheromone Values

```
1: i ← 1
2: while i <= noOfTargets do
3:    k ← 1
4:    while k <= noOfWeapons do
5:       if iterationBestSolutionAllocations[k] == i then
6:          pheromoneValues[i][k] ← pheromoneValues[i][k] × (1 −
            pheromoneIncreaseRate)
7:          pheromoneValues[i][k] ← pheromoneValues[i][k] +
            (pheromoneIncreaseRate × (1/bestSolutionValue))
8:       end if
9:       k ← k + 1
10:   end while
11:   i ← i + 1
12: end while
```

3.4.4 Particle Swarm Optimization Algorithm

Particle swarm optimization takes inspiration from the movement and intelligence of swarms.

In particle swarm optimization, particles form a swarm and the position of a particle corresponds to a candidate solution to the problem. Particles move around the search space according to a simple formula over the particle's position (given as Eqn. 3.7) and velocity (given as Eqn. 3.8). Initially the positions and velocities of all particles are given randomly. Then, objective function value is calculated using the particles' positions (candidate solution). If the optimization criteria are met, the algorithm returns the best particle. Otherwise, the positions

and velocities are updated based on the objective function value, and objective function value is recalculated using the new positions.

Each particle's velocity is updated according to the formula given in Eqn. 3.7:

$$\vec{v_j}^{t+1} = \omega \vec{v_j}^{t} + \varphi_b \vec{r_b}^{t} (\vec{b_j}^{t} - \vec{x_j}^{t}) + \varphi_g \vec{r_g}^{t} (\vec{g}^{t} - \vec{x_j}^{t}) \tag{3.7}$$

Each particle's position is updated according to the formula given in Eqn. 3.8:

$$\vec{x_j}^{t+1} = \vec{x_j}^{t} + \vec{v_j}^{t+1} \tag{3.8}$$

$\vec{x_j}^{t}$ is the position of particle j at time t.

$\vec{v_j}^{t}$ is the velocity of particle j at time t.

$\vec{b_j}^{t}$ is the personal best position of particle j at time t.

$\vec{g}$ is the global best position of the swarm.

ω is the constant to represent the importance of the previous velocity vector *(momentum)*. In the implementation, this constant is set to 0.8 for simplicity.

φ_b and φ_g is the constants to represent the importance of personal best position *(cognitive component)* and global best position *(social component)*, respectively. In the implementation, both constants are set to 0.2 for simplicity.

$\vec{r_b}^{t}$ and $\vec{r_g}^{t}$ are vectors with random numbers uniformly distributed between 0 and 1.

A sample flow chart of the algorithm is given in Figure 3.7.

The pseudocode of the implemented Particle Swarm Optimization algorithm is given as Algorithm 25.

In the implemented particle swarm optimization algorithm, first the initial swarm is generated from particles *(5)*. Random positions and random velocities are assigned to each particle. The generation of the initial swarm is given as Algorithm 26.

Then, for each particle *(8 - 23)*, the objective function value is calculated based

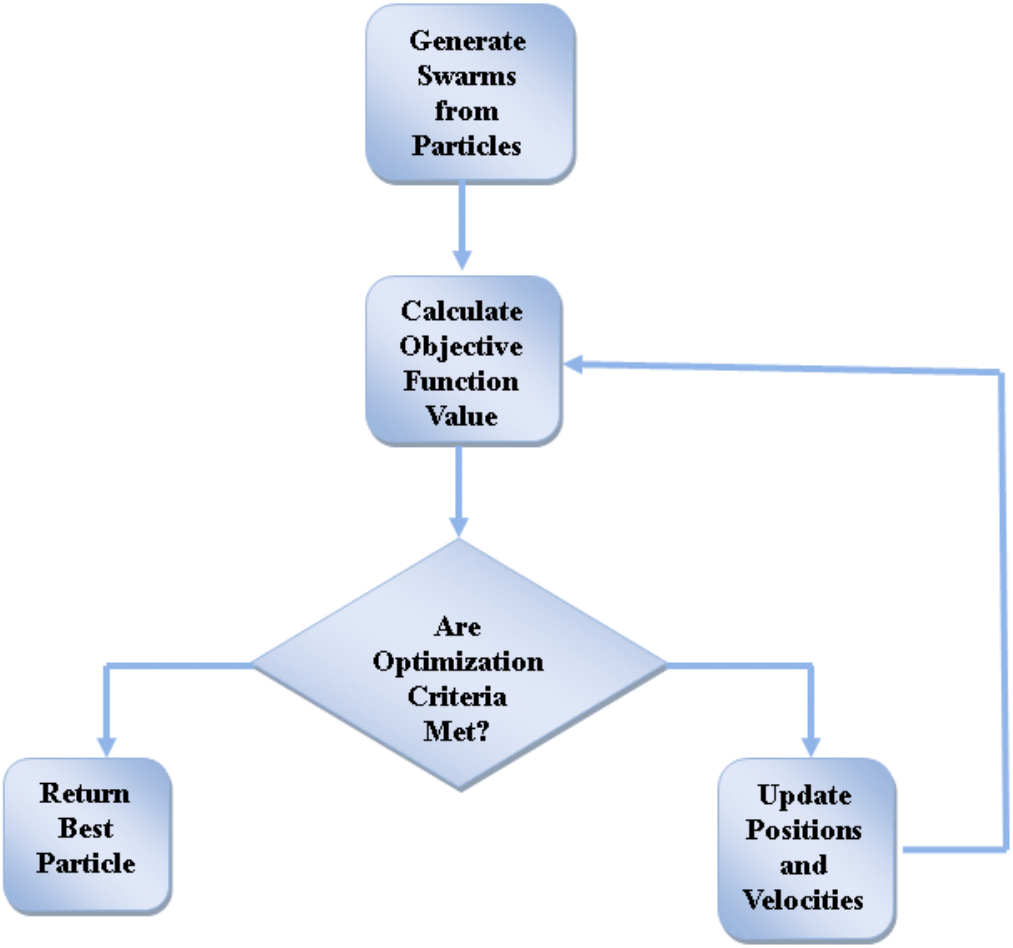

Figure 3.7: Flow Chart of Particle Swarm Optimization Algorithm

Algorithm 25 Particle Swarm Optimization

```
1: startTime ← Now
2: endTime ← startTime + allowedSearchTime
3: solution.allocations ← {}
4: solution.value ← MaxValue
5: swarm ← GenerateInitialSwarm(noOfParticles)
6: while endTime < Now do
7:    i ← 1
8:    while i <= noOfParticles do
9:       swarm[i].solutionValue ← CalculateSolValue(swarm[i].allocations)
10:      if swarm[i].solutionValue == solution.solutionValue then
11:         swarm[i].velocities ← GetRandomVelocities()
12:      else
13:         if swarm[i].solutionValue < swarm[i].bestSolutionValue then
14:            swarm[i].bestSolutionValue ← swarm[i].solutionValue
15:            swarm[i].bestAllocations ← swarm[i].allocations
16:            if swarm[i].solutionValue < solution.solutionValue then
17:               solution.allocations ← swarm[i].allocations
18:               solution.solutionValue ← swarm[i].solutionValue
19:            end if
20:         end if
21:      end if
22:      i ← i + 1
23:   end while
24:   i ← 1
25:   while i <= noOfParticles do
26:      k ← 1
27:      while k <= noOfWeapons do
28:         swarm[i].velocities[k] ← UpdateVelocity(swarm[i], k)
29:         swarm[i].allocations[k] ← UpdatePosition(swarm[i], k)
30:         k ← k + 1
31:      end while
32:      i ← i + 1
33:   end while
34: end while
35: return solution
```

Algorithm 26 Generate Initial Swarm

$swarm \leftarrow \{\}$

$i \leftarrow 1$

while $i <= noOfParticles$ **do**

$particle.allocations \leftarrow GetRandomPositions()$

$particle.bestAllocations \leftarrow particle.allocations$

$particle.velocities \leftarrow GetRandomVelocities()$

$particle.solutionValue \leftarrow MaxValue$

$particle.bestSolutionValue \leftarrow MaxValue$

$swarm.add(particle)$

$i \leftarrow i + 1$

end while

return $swarm$

on the particle's position or allocations *(9)*. If the objective function value of the particle is equal to the global solution value *(10)*, then the velocities of the particles are updated randomly *(11)*. Otherwise, if the solution value of the particle is better (less) than the global solution value *(16)*, then the swarm's best known position (global solution) is updated to the particle's position or allocations *(17, 18)*. Next, the position and the velocity of all particles in the swarm is updated *(25 - 33)*.

The velocity of a particle is updated according to Eqn. 3.7, the pseudocode of which is given in Algorithm 27.

Algorithm 27 Update Velocity

$velocity \leftarrow particle.velocities[k] \times momentum$

$velocity \leftarrow velocity + random \times cognitiveConstant \times (particle.bestAllocations[k] - particle.allocations[k])$

$velocity \leftarrow velocity + random \times socialConstant \times (solution.allocations[k] - particle.allocations[k])$

return $velocity$

The position of a particle is updated according to Eqn. 3.8, the pseudocode of which is given in Algorithm 28.

Algorithm 28 Update Position

position ← *particle.allocations*[*k*] + *particle.velocities*[*k*]

return *position*

This whole process is repeated until the maximum allowed search time is reached and the swarm's best known position (global solution) is returned.

3.4.5 Particle Swarm Optimization Algorithm Improved with MMR Algorithms

Instead of generating the initial swarm by assigning positions to the particles randomly, using an MMR algorithm in the generation phase improves the solution. Similar to the improved version of the genetic algorithm, *Greedy MMR* and *Advanced MMR* algorithms are used.

The pseudocodes of the methods that use Greedy MMR algorithm and Advanced MMR algorithm are given in Algorithms 29 and 30, respectively.

In the generation of the initial swarm, first particle is chosen to be the solution found by applying the Greedy/Advanced MMR. And the other particles are chosen by swapping allocations of that solution. By doing so instead of generating the initial swarm by assign random positions to the particles, better individuals are created and the solution is closer to the optimal solution.

Note that, since evolutionary algorithms use randomization, all evolutionary algorithms are nondeterministic algorithms.

3.5 Bipartite Graph Matching Algorithms

Before going into the details of the bipartite graph matching algorithms, it is necessary to give a brief background information about bipartite graphs. The definitions given in this section are taken from the paper referenced in [23].

A graph $G = (V, E)$ is **bipartite** if its vertices V can be divided into two disjoint sets, V^+ and V^-, such that each edge connects a vertex in V^+ to one in V^-.

Algorithm 29 Generate Initial Swarm using Greedy MMR

$swarm \leftarrow \{\}$
$mmrSol \leftarrow ApplyGreedyMmrAlgorithm()$
$particle.allocations \leftarrow mmrSol.allocations$
$particle.bestAllocations \leftarrow mmrSol.allocations$
$particle.solutionValue \leftarrow mmrSol.solutionValue$
$particle.bestSolutionValue \leftarrow mmrSol.solutionValue$
$particle.velocities \leftarrow GetRandomVelocities()$
$swarm.Add(particle)$
$i \leftarrow 1$
while $i < noOfParticles$ **do**
  $neighborAlloc \leftarrow RandomlySwapAllocations(mmrSol.allocations)$
  $neighborSolVal \leftarrow CalculateSolutionValue(neighborAlloc)$
  $particle.allocations \leftarrow neighborAlloc$
  $particle.bestAllocations \leftarrow neighborAlloc$
  $particle.solutionValue \leftarrow neighborSolVal$
  $particle.bestSolutionValue \leftarrow neighborSolVal$
  $particle.velocities \leftarrow GetRandomVelocities()$
  $swarm.add(particle)$
  $i \leftarrow i + 1$
end while
return $swarm$

Algorithm 30 Generate Initial Swarm using Advanced MMR

$swarm \leftarrow \{\}$
$mmrSol \leftarrow ApplyAdvancedMmrAlgorithm()$
$particle.allocations \leftarrow mmrSol.allocations$
$particle.bestAllocations \leftarrow mmrSol.allocations$
$particle.solutionValue \leftarrow mmrSol.solutionValue$
$particle.bestSolutionValue \leftarrow mmrSol.solutionValue$
$particle.velocities \leftarrow GetRandomVelocities()$
$swarm.Add(particle)$
$i \leftarrow 1$
while $i < noOfParticles$ **do**
  $neighborAlloc \leftarrow RandomlySwapAllocations(mmrSol.allocations)$
  $neighborSolVal \leftarrow CalculateSolutionValue(neighborAlloc)$
  $particle.allocations \leftarrow neighborAlloc$
  $particle.bestAllocations \leftarrow neighborAlloc$
  $particle.solutionValue \leftarrow neighborSolVal$
  $particle.bestSolutionValue \leftarrow neighborSolVal$
  $particle.velocities \leftarrow GetRandomVelocities()$
  $swarm.add(particle)$
  $i \leftarrow i + 1$
end while
return $swarm$

A subset M of E is a **matching** if no vertex is incident to more than one edge in M.

Maximum matching is a matching that contains the largest possible number of edges.

Alternating path is a path in which the edges belong alternatively to the matching and not the matching.

Augmenting path is an alternating path that starts from and ends on unmatched vertices.

In the general form of the assignment problem, there are a number of workers and a number of jobs. Any worker can be assigned to perform any job, incurring some cost that may vary depending on the worker-job assignment. It is required to perform all jobs by assigning exactly one worker to each job in such a way that total cost of the assignment is minimized.

The assignment problem consists of finding a maximum weight matching in a weighted bipartite graph. The Munkres' Assignment Algorithm (also known as Hungarian Algorithm or Kuhn-Munkres Algorithm) is one of many algorithms that have been devised to solve the assignment problem within polynomial time.

To better understand the algorithm details, the matrix representation of the graph is used. As an example, assume that there are 4 workers (w_1, w_2, w_3 and w_4) and 4 jobs (j_1, j_2, j_3 and j_4) and the cost matrix is:

$$\begin{pmatrix} 3 & 4 & 7 & 1 \\ 4 & 5 & 2 & 2 \\ 2 & 6 & 8 & 6 \\ 6 & 2 & 3 & 1 \end{pmatrix}$$

This matrix can be interpreted as the cost of worker w_1 performing job j_3 is 7, the cost of worker w_3 performing job j_3 is 8, etc. If the assignment is (w_1, j_3), (w_2, j_1), (w_3, j_4), (w_4, j_2), then the cost is $7 + 4 + 6 + 2 = 19$. The assignment algorithms try to minimize this total cost.

3.5.1 Munkres' Assignment Algorithm

In this section, the basic Munkres' Assignment Algorithm to solve Weapon - Target Allocation problem is described.

Although in the general assignment problem exactly one worker should be assigned to each job, in the Weapon - Target Allocation problem, more than one weapon can be assigned to a single target. In other words, there is no one-to-one correspondence between workers and jobs in the Weapon - Target Allocation problem.

The pseudocode of the Munkres' Assignment Algorithm is given in Algorithm 31.

As it can be seen from the pseudocode, Munkres' Assignment Algorithm has 8 steps:

1. **Constructing the Cost Matrix:**

   To be able to apply Munkres' Assignment Algorithm, the cost matrix needs to be square. In the cost matrix construction step, an NxN matrix is created where N is the maximum of $|W|$ and $|T|$.

   The value of each cell (k, i) in the cost matrix is given according to the equation 3.9:

   $$CostMatrix[k][i] = targetValues[i] \times (1 - killProbabilities[i][k]) \quad (3.9)$$

   If $|W|$ is greater than $|T|$, the cells of the dummy *columns* contain the maximum value of the matrix. If $|T|$ is greater than $|W|$, the cells of the dummy *rows* contain the maximum value of the matrix.

   The pseudocode of the method for constructing the cost matrix is given in Algorithm 32.

   After this step, the algorithm goes into Step-2 which is the step of reducing rows.

2. **Reducing the Rows:**

Algorithm 31 Munkres' Assignment Algorithm

```
1: step ← 1
2: DONE ← false
3: while NOTDONE do
4:    if step == STEP1 then
5:       step ← ConstructCostMatrix()
6:    end if
7:    if step == STEP2 then
8:       step ← ReduceRows()
9:    end if
10:   if step == STEP3 then
11:      step ← StarZeros()
12:   end if
13:   if step == STEP4 then
14:      step ← CoverColumns()
15:   end if
16:   if step == STEP5 then
17:      step ← PrimeZeros()
18:   end if
19:   if step == STEP6 then
20:      step ← ConstructAlternatingZeros()
21:   end if
22:   if step == STEP7 then
23:      step ← AddSubtractMinValue()
24:   end if
25:   if step == STEP8 then
26:      solution ← GetSolution()
27:      DONE ← true
28:   end if
29: end while
30: return solution
```

Algorithm 32 Constructing the Cost Matrix

$InitializeWithZeros(costMatrix)$
if $noOfWeapons > noOfTargets$ **then**
  $n \leftarrow noOfWeapons$
else
  $n \leftarrow noOfTargets$
end if
$k \leftarrow 1$
while $k <= n$ **do**
  $i \leftarrow 1$
  while $i <= n$ **do**
    $costMatrix[k][i] \leftarrow targetValues[i] \times (1 - killProbabilities[i][k])$
    $i \leftarrow i + 1$
  end while
  $k \leftarrow k + 1$
end while
$k \leftarrow 1$
while $k <= n$ **do**
  $i \leftarrow 1$
  while $i <= n$ **do**
    if $costMatrix[k][i] == 0$ **then**
      $costMatrix[k][i] \leftarrow maxValueInMatrix(costMatrix)$
    end if
    $i \leftarrow i + 1$
  end while
  $k \leftarrow k + 1$
end while
return $STEP2$

In the row reducing step, for each row of the matrix, the smallest element of that row is found and subtracted from every element in that row.

The pseudocode of the method for reducing the rows of the matrix is given in Algorithm 33.

Algorithm 33 Reducing the Rows

```
k ← 1
while k <= n do
  minValueOfRow ← MaxValue
  i ← 1
  while i <= n do
    if costMatrix[k][i] < minValueOfRow then
      minValueOfRow ← costMatrix[k][i]
    end if
    i ← i + 1
  end while
  i ← 1
  while i <= n do
    costMatrix[k][i] ← costMatrix[k][i] − minValueOfRow
    i ← i + 1
  end while
  k ← k + 1
end while
return STEP3
```

After this step, the algorithm goes into Step-3 which is the step of starring the zeros.

3. **Starring the Zeros:**

   In the starring zeros step, first a zero is found in the resulting matrix. Then, if there is no starred zero in the row or column of that zero, the found zero is starred. This process is repeated for each element in the matrix.

   The pseudocode of the method for starring the zeros in the matrix is given

in Algorithm 34.

Algorithm 34 Starring the Zeros

```
k ← 1
while k <= n do
  i ← 1
  while i <= n do
    if costMatrix[k][i] == 0 AND !rowCover[k] AND !columnCover[i]
    then
      marks[k][i] ← Starred
      rowCover[k] ← true
      columnCover[i] ← true
    end if
    i ← i + 1
  end while
  k ← k + 1
end while
return STEP4
```

After this step, the algorithm goes into Step-4 which is the step of covering the columns of the matrix.

4. **Covering the Columns:**

   In the covering columns step, each column containing a starred zero is covered. If all of the columns are covered, this means that the starred zeros describe a complete set of unique assignments.

   The pseudocode of the method for covering the columns in the matrix is given in Algorithm 35.

   After this step, if all of the columns in the matrix are covered, the algorithm goes into Step-8, otherwise it goes into Step-5.

5. **Priming Zeros:**

   In the priming zeros step, first an uncovered zero is found and primed. If there is no starred zero in the row containing the primed zero, the algorithm continues with Step-6. Otherwise, the row containing the primed

Algorithm 35 Covering the Columns

```
k ← 1
while k <= n do
  i ← 1
  while i <= n do
    if marks[k][i] == Starred then
      columnCover[i] ← true
    end if
    i ← i + 1
  end while
  k ← k + 1
end while
i ← 1
coveredColumnsCount ← 0
while i <= n do
  if columnCover[i] == true then
    coveredColumnsCount ← coveredColumnsCount + 1
  end if
  i ← i + 1
end while
if coveredColumnsCount == n then
  return STEP8
else
  return STEP5
end if
```

zero is covered and the column containing the starred zero is uncovered. This process is repeated until there are no uncovered zeros left. If there are no uncovered zeros left, the algorithm continues with Step-7.

The pseudocode of the method for priming the zeros in the matrix is given in Algorithm 36.

Algorithm 36 Priming Zeros

```
while true do
  position ← FindAZeroPosition(costMatrix)
  if position == NULL then
    return STEP7
  end if
  marks[position.row][position.column] ← Primed
  columnOfStarInRow ← FindColumnOfStarInRow(position.row)
  if columnOfStarInRow == NULL then
    uncoveredPrimedZeroPosition ← position
    return STEP6
  end if
  position.column ← columnOfStarInRow
  rowCover[position.row] ← true
  columnCover[position.column] ← false
end while
```

6. **Constructing Alternating Zeros:**

   In the alternating zeros construction step, a series of alternating primed and starred zeros is constructed. $Z0$ is the uncovered primed zero found in Step-5. $Z1$ is the starred zero in the column of $Z0$ and $Z2$ is the primed zero in the row of $Z1$. This alternation process is repeated until the series terminate at a primed zero that has no starred zero in its column. Then, the path is augmented by unstarring each starred zero of the series, starring each primed zero of the series, erasing all primes and uncovering every line in the matrix.

   The pseudocode of the method for constructing the alternating zeros is given in Algorithm 37.

Algorithm 37 Constructing Alternating Zeros

$Z0 \leftarrow uncoveredPrimedZeroPosition$

$path \leftarrow \{\}$

while $true$ **do**

$path.Add(Z0)$

$row \leftarrow FindRowOfStarInColumn(Z0.column)$

if $row == NULL$ **then**

$BREAK$

end if

$Z1.row \leftarrow row$

$Z1.column \leftarrow Z0.column$

$path.Add(Z1)$

$Z0.row \leftarrow Z1.row$

$Z0.column \leftarrow FindColumnOfPrimeZeroInRow(Z1.row)$

end while

$AugmentPath(path)$

$ClearCovers()$

$ErasePrimes()$

return $STEP4$

After this step, the algorithm again goes into Step-4 which is the step of covering the columns of the matrix.

7. **Adding/Subtracting Minimum Uncovered Value of the Matrix:**

   In this step, the minimum uncovered value of the matrix is added to every element of each covered row and it is subtracted from every element of each uncovered column.

   The pseudocode of the method for adding/subtracting the minimum value of the matrix is given in Algorithm 38.

Algorithm 38 Adding/Subtracting Minimum Uncovered Value of the Matrix

```
minValueOfMatrix ← FindMinValOfUncoveredElements()
k ← 1
while k <= n do
  i ← 1
  while i <= n do
    if rowCover[k] == true then
      costMatrix[k][i] ← costMatrix[k][i] + minValueOfMatrix
    end if
    if columnCover[i] == false then
      costMatrix[k][i] ← costMatrix[k][i] − minValueOfMatrix
    end if
    i ← i + 1
  end while
  k ← k + 1
end while
return STEP5
```

After this step, the algorithm goes into Step-5 which is the step of priming zeros.

8. **Getting the Solution:**

   The cells containing the starred zeros construct the solution. If the number of targets ($|T|$) is equal to the number of weapons ($|W|$), then the

solution is optimal. However, most of the times, these two values are not equal. If $|T|$ is greater than $|W|$, the starred zeros in the dummy rows are discarded and the other starred zeros form the solution since all weapons are allocated to a target. On the other hand, if $|T|$ is less than $|W|$, the starred zeros in the dummy columns cannot be discarded because if they are discarded, there will be unallocated weapons which is not desired. So, for these weapons that are allocated to dummy targets, new real targets are assigned.

The pseudocode of the method for getting the solution is given in Algorithm 39.

Algorithm 39 Getting the Solution

```
solution.allocations ← {}
solution.solutionValue ← MaxValue
starredZeros ← GetStarredZeros()
k ← 1
while k <= noOfWeapons do
  starredZeroPosition ← starredZeros[k]
  if starredZeroPosition.column > noOfTargets then
    targetIndex ← FindOptimalTargetForWeapon(k)
  else
    targetIndex ← starredZeroPosition.column
  end if
  solution.allocations.Add(targetIndex)
  k ← k + 1
end while
solution.solutionValue ← CalculateSolutionValue(solution.allocations)
return solution
```

The algorithm terminates by returning the solution.

An Example Application of Munkres' Assignment Algorithm:

To be able to follow the steps easily, the application of steps are described with the following sample problem instance: $T = 4$, $W = 4$, V = (0.95 0.75 0.75 0.69)

and

$$P = \begin{pmatrix} 0.72 & 0.96 & 0.66 & 0.51 \\ 0.67 & 0.98 & 0.95 & 0.68 \\ 0.53 & 0.68 & 0.96 & 0.53 \\ 0.88 & 0.88 & 0.64 & 0.84 \end{pmatrix}$$

Step-1: Construct the Cost Matrix

Applying this step to the example, the cost matrix becomes

$$C = \begin{pmatrix} 0.26 & 0.25 & 0.36 & 0.08 \\ 0.04 & 0.01 & 0.24 & 0.08 \\ 0.32 & 0.04 & 0.03 & 0.25 \\ 0.47 & 0.24 & 0.35 & 0.11 \end{pmatrix}$$

The algorithm continues with Step-2.

Step-2: Reduce Rows

The minimum values in the rows are 0.08, 0.01, 0.03 and 0.11. Subtracting these values from the values of rows, the cost matrix becomes

$$C = \begin{pmatrix} 0.18 & 0.17 & 0.28 & 0 \\ 0.03 & 0 & 0.23 & 0.07 \\ 0.29 & 0.01 & 0 & 0.22 \\ 0.36 & 0.13 & 0.24 & 0 \end{pmatrix}$$

The algorithm continues with Step-3.

Step-3: Star the Zeros

After starring the zeros, the cost matrix becomes

$$C = \begin{pmatrix} 0.18 & 0.17 & 0.28 & 0* \\ 0.03 & 0* & 0.23 & 0.07 \\ 0.29 & 0.01 & 0* & 0.22 \\ 0.36 & 0.13 & 0.24 & 0 \end{pmatrix}$$

Step-4: Cover the Columns

Applying the step of covering the columns to the example, the columns except the first column are covered. Since all columns are not covered, the algorithm continues with Step-5 for the example.

Step-5: Prime the Zeros

For the example, all zeros are covered. Since there is no uncovered zero, no zeros are marked as primed and algorithm continues with Step-7.

Step-7: Add/Subtract the Minimum Uncovered Value of the Matrix

There is no row covered in the resulting matrix and the only uncovered column is the first column. The smallest value of the uncovered elements is 0.03. So, this value is subtracted from each element of the first column. Then, the resulting matrix becomes

$$C = \begin{pmatrix} 0.15 & 0.17 & 0.28 & 0* \\ 0 & 0* & 0.23 & 0.07 \\ 0.26 & 0.01 & 0* & 0.22 \\ 0.33 & 0.13 & 0.24 & 0 \end{pmatrix}$$

The algorithm continues with Step-5.

Step-5: Prime the Zeros

The only uncovered zero is located at the cell $(2, 1)$. This zero is primed. Since there is a starred zero at the row of this primed zero, this row (row 2) is covered and the column containing the starred zero (column 2) is uncovered. Since, there are no more uncovered zeros, the algorithm continues with Step-7.

The cost matrix after applying this step becomes

$$C = \begin{pmatrix} 0.15 & 0.17 & 0.28 & 0* \\ 0' & 0* & 0.23 & 0.07 \\ 0.26 & 0.01 & 0* & 0.22 \\ 0.33 & 0.13 & 0.24 & 0 \end{pmatrix}$$

Step-7: Add/Subtract the Minimum Uncovered Value of the Matrix

The only covered row is 2. row and the uncovered columns are 1^{st} and 2^{nd} columns. The smallest value among the uncovered elements is 0.01. So, this value is added to the elements of second row and subtracted from the elements of 1^{st} and 2^{nd} column. Then, the resulting matrix becomes

$$C = \begin{pmatrix} 0.14 & 0.16 & 0.28 & 0* \\ 0' & 0* & 0.24 & 0.08 \\ 0.25 & 0 & 0* & 0.22 \\ 0.32 & 0.12 & 0.24 & 0 \end{pmatrix}$$

The algorithm continues with Step-5.

Step-5: Prime the Zeros

The only uncovered zero is located at the cell $(3, 2)$. This zero is primed. Since there is a starred zero at the row of this primed zero, this row (row 3) is covered and the column containing the starred zero (column 3) is uncovered. Since, there are no more uncovered zeros, the algorithm continues with Step-7.

The cost matrix after applying this step becomes

$$C = \begin{pmatrix} 0.14 & 0.16 & 0.28 & 0* \\ 0' & 0* & 0.24 & 0.08 \\ 0.25 & 0' & 0* & 0.22 \\ 0.32 & 0.12 & 0.24 & 0 \end{pmatrix}$$

Step-7: Add/Subtract the Minimum Uncovered Value of the Matrix

The covered rows are 2^{nd} and 3^{rd} rows and the uncovered columns are 1^{st}, 2^{nd} and 3^{rd} columns. The smallest value among the uncovered elements is 0.12. So, this value is added to the elements of 2^{nd} and 3^{rd} rows and subtracted from the elements of 1^{st}, 2^{nd} and 3^{rd} columns. Then, the resulting matrix becomes

$$C = \begin{pmatrix} 0.02 & 0.04 & 0.16 & 0* \\ 0' & 0* & 0.23 & 0.19 \\ 0.25 & 0' & 0* & 0.34 \\ 0.2 & 0 & 0.12 & 0 \end{pmatrix}$$

The algorithm continues with Step-5.

Step-5: Prime the Zeros

The only uncovered zero is located at the cell $(4, 2)$. This zero is primed. Since there is no starred zero at the row of this primed zero, the algorithm continues with Step-6.

The cost matrix after applying this step becomes

$$C = \begin{pmatrix} 0.02 & 0.04 & 0.16 & 0* \\ 0' & 0* & 0.23 & 0.19 \\ 0.25 & 0' & 0* & 0.34 \\ 0.2 & 0' & 0.12 & 0 \end{pmatrix}$$

Step-6: Construct Alternating Zeros:

$Z0$ is the uncovered primed zero located at the cell $(4, 2)$. $Z1$ is the starred zero located at the cell $(2, 2)$ and $Z2$ is the primed zero located at the cell $(2, 1)$. Since there is no starred zero at the column of $Z2$ alternating process is terminated. So, the path consists of the cells $(4, 2)$, $(2, 2)$ and $(2, 1)$. Then, this path is augmented by unstarring the starred zeros and starring the primed zeros. Next, all primes are cleared and all lines are uncovered.

The cost matrix after applying this step becomes

$$C = \begin{pmatrix} 0.02 & 0.04 & 0.16 & 0* \\ 0* & 0 & 0.23 & 0.19 \\ 0.25 & 0 & 0* & 0.34 \\ 0.2 & 0* & 0.12 & 0 \end{pmatrix}$$

The algorithm continues with Step-4.

Step-4: Cover the Columns

Applying the step of covering the columns to the resulting matrix, this time all columns are covered. So, the algorithm goes into Step-8.

Step-8: Get the Solution

The cells containing the starred zeros construct the solution. So, the solution is as follows:
1^{st} weapon is allocated for 4^{th} target
2^{nd} weapon is allocated for 1^{st} target
3^{rd} weapon is allocated for 3^{rd} target
4^{th} weapon is allocated for 2^{nd} target

Note that, if $|W| = |T|$, then the Munkres' Assignment Algorithm gives the optimal solution in polynomial time.

3.5.2 Munkres Assignment Algorithm Improved with MMR Algorithms

If $|T|$ is less than $|W|$, the getting solution process in Step-8 can be improved by making use of MMR algorithms. For the weapons that are unallocated (or allocated to dummy targets) and the targets, the problem is reconstructed. Then, *greedy MMR algorithm improved with local search* (given in Section 3.3.2) or *advanced MMR algorithm improved with local search* (given in Section 3.3.5) is applied to the subproblem. The solution is used for the allocation of unallocated weapons to real targets. This process improves the solution, the results are given in Chapter 4.

3.5.3 Advanced Munkres Assignment Algorithm

The algorithm given in this section is a combination of the other Munkres' assignment algorithms given in previous sections. Basically, if $|T|$ is equal to $|W|$, it runs basic Munkres' assignment algorithm given in section 3.5.1 and returns the solution. If $|T|$ is less than $|W|$, then it runs the improved algorithms given in section 3.5.2 and returns the solution with best objective function value. Otherwise, if time is enough, the problem is divided into subproblems and Munkres' assignment algorithm is applied to the subproblems and the solution with the best objective function value is returned.

3.6 A Suggested Algorithm

By making use of the results of all algorithms described in this chapter, an algorithm is proposed (Algorithm 40).

Algorithm 40 Suggested Algorithm

1: $exhaustiveSol \leftarrow ApplyExhaustiveSearchAlgorithm()$
2: **if** $exhaustiveSol == NOTNULL$ **then**
3: **return** $exhaustiveSol$
4: **end if**
5: $advancedMmrSol \leftarrow ApplyAdvancedMmrAlgorithm()$
6: $advancedMunkresSol \leftarrow ApplyAdvancedMunkresAlgorithm()$
7: **if** $advancedMmrSol.solutionValue < advancedMunkresSol.solutionValue$ **then**
8: **return** $advancedMmrSol$
9: **else**
10: **return** $advancedMunkresSol$
11: **end if**

This proposed algorithm first tries to apply exhaustive search algorithm since it gives the optimal solution. If the maximum allowed time is enough for finding the optimal solution, then the algorithm returns that solution *(1 - 3)*. Otherwise, it applies both Advanced MMR algorithm *(5)* and Advanced Munkres algorithm *(6)* and returns the better solution *(7 - 10)*.

CHAPTER 4

EVALUATION OF ALGORITHMS

Although in the literature, the study on the evaluation of the Weapon - Target Allocation algorithms is very sparse, the results obtained from the evaluation of the algorithms are very crucial because of the critical consequences. In this chapter, the algorithms described in the previous chapter are evaluated in terms of performance and optimality metrics. The following algorithms are evaluated:

1. Exhaustive Search Algorithm
2. Random Search Algorithm
3. Greedy MMR Algorithm
4. Greedy MMR Algorithm Improved with Local Search
5. Random MMR Algorithm
6. Advanced MMR Algorithm
7. Advanced MMR Algorithm Improved with Local Search
8. Genetic Algorithm
9. Genetic Algorithm Improved with Greedy MMR Algorithm
10. Genetic Algorithm Improved with Advanced MMR Algorithm
11. Ant Colony Optimization Algorithm
12. Particle Swarm Optimization Algorithm

13. Particle Swarm Optimization Algorithm Improved with Greedy MMR Algorithm
14. Particle Swarm Optimization Algorithm Improved with Advanced MMR Algorithm
15. Munkres' Assignment Algorithm
16. Munkres' Assignment Algorithm Improved with Greedy MMR Algorithm
17. Munkres' Assignment Algorithm Improved with Advanced MMR Algorithm
18. Advanced Munkres' Assignment Algorithm
19. Suggested Algorithm

In the following subsections, first, the testbed that is developed for the evaluation of algorithms is introduced. Next, the experimental results that are obtained by applying the algorithms to sample problem instances are explained.

4.1 Testbed Developed for the Evaluation of Algorithms

In this subsection, the testbed that has been developed for comparing the algorithms is introduced. The testbed has been implemented in CSharp programming language and Microsoft Visual Studio 2010 has been used as the development environment.

Below, the requirements and design model of the testbed is given.

4.1.1 Requirements of the Testbed

A testbed that is developed for comparing the Weapon - Target Allocation algorithms should:

1. Create the problem instance by taking the inputs (number of weapons and number of targets) from the user.

2. Restrict the running time of the algorithm so that maximum allowed time which is set by the user is not exceeded.

3. Allow user to select the algorithms to be compared.

4. Apply all selected algorithms on the **same** problem instance.

5. Be able to save the problem instance (number of weapons, number of targets, target values, kill probabilities, etc.) and comparison results (solution value, execution time, the allocations and order for each algorithm that is applied) for future use.

6. Order the selected algorithms by the solution value.

7. Allow users to develop new algorithms and compare them to the existing algorithms easily.

4.1.2 Design Model of the Testbed

The implemented testbed consists of basically 3 packages: Algorithm Package, Scenario Package and Graphical User Interface (GUI) Package. The diagrams given in this section are prepared by making use of the Enterprise Architect 7.5 tool.

1. **Algorithm Package**

   The classes and packages of the Algorithm Package is shown in Figure 4.1. This package includes the **base class** *(CBaseAlgorithm)* with a method for calculating the objective function value and an abstract method for finding the solution, and the **algorithm packages**. Algorithm packages include:

   - Search Algorithms (Shown in Figure 4.2),
   - Maximum Marginal Return Algorithms (Shown in Figure 4.3),
   - Evolutionary Algorithms (Shown in Figure 4.4),
   - Bipartite Graph Matching Algorithms (Shown in Figure 4.5),

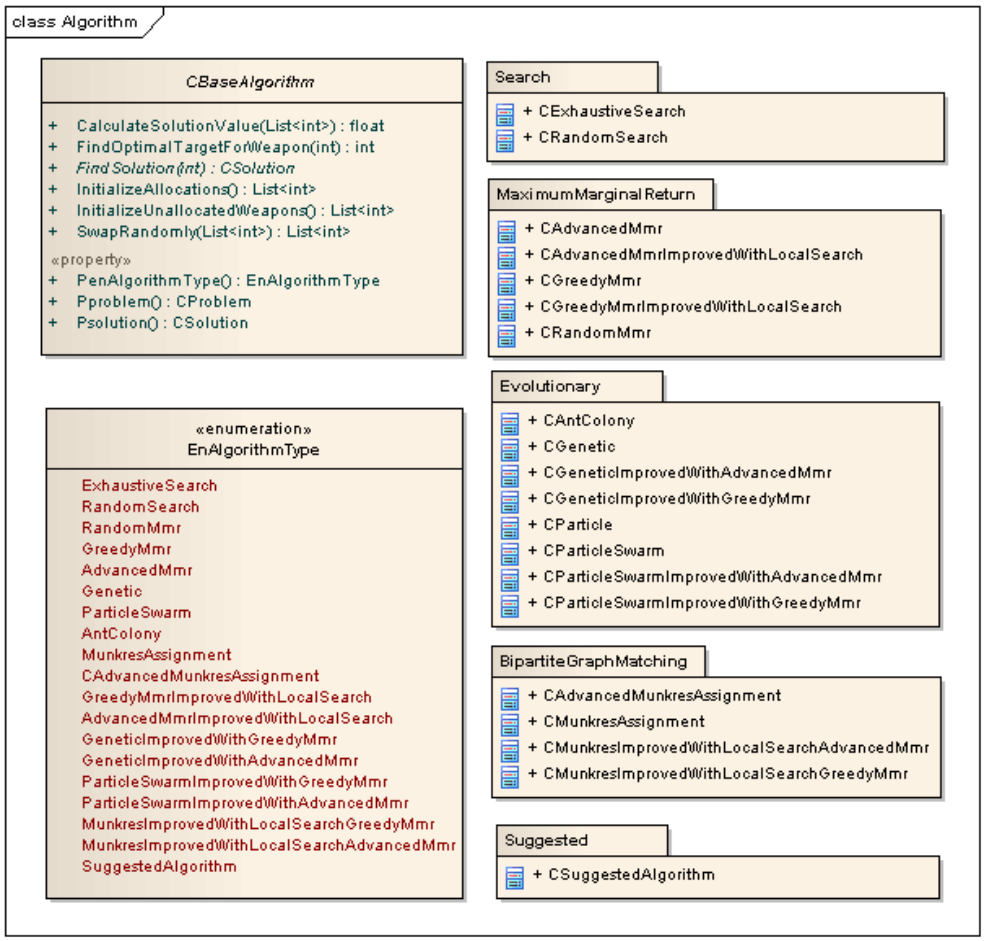

Figure 4.1: Classes and Packages of Algorithm Package

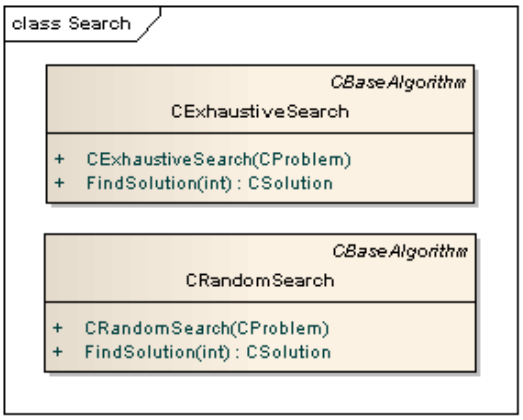

Figure 4.2: Search Algorithms Package

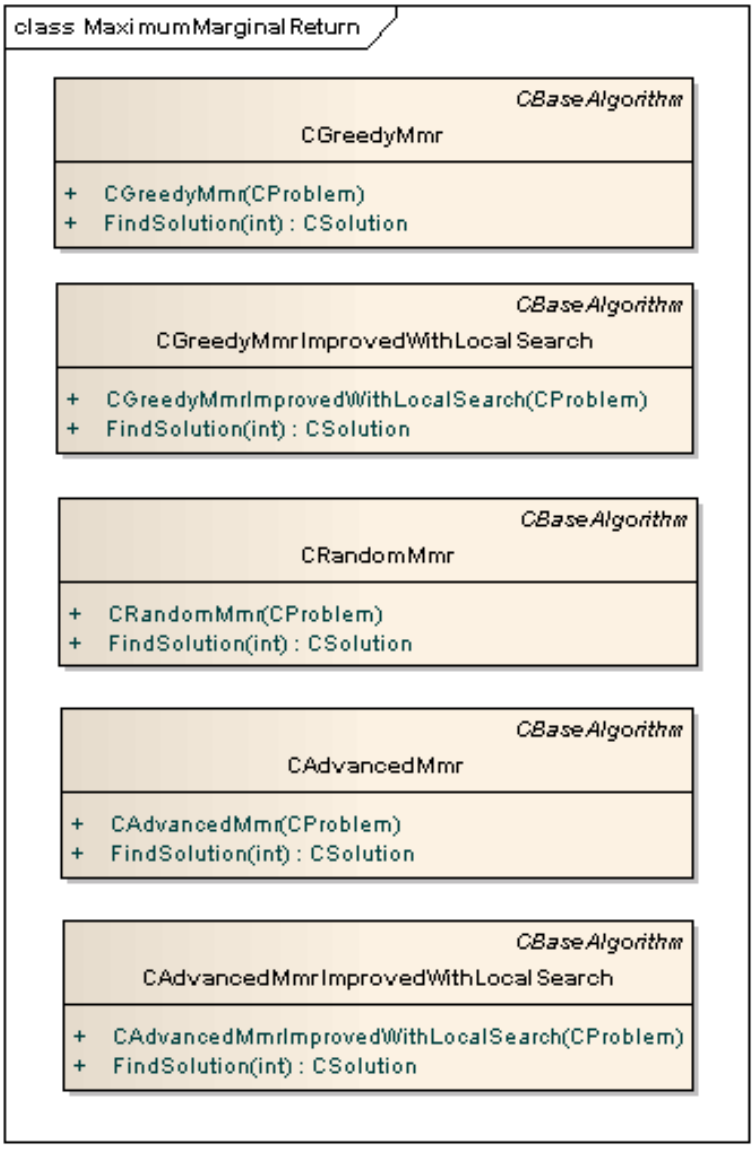

Figure 4.3: Maximum Marginal Return Algorithms Package

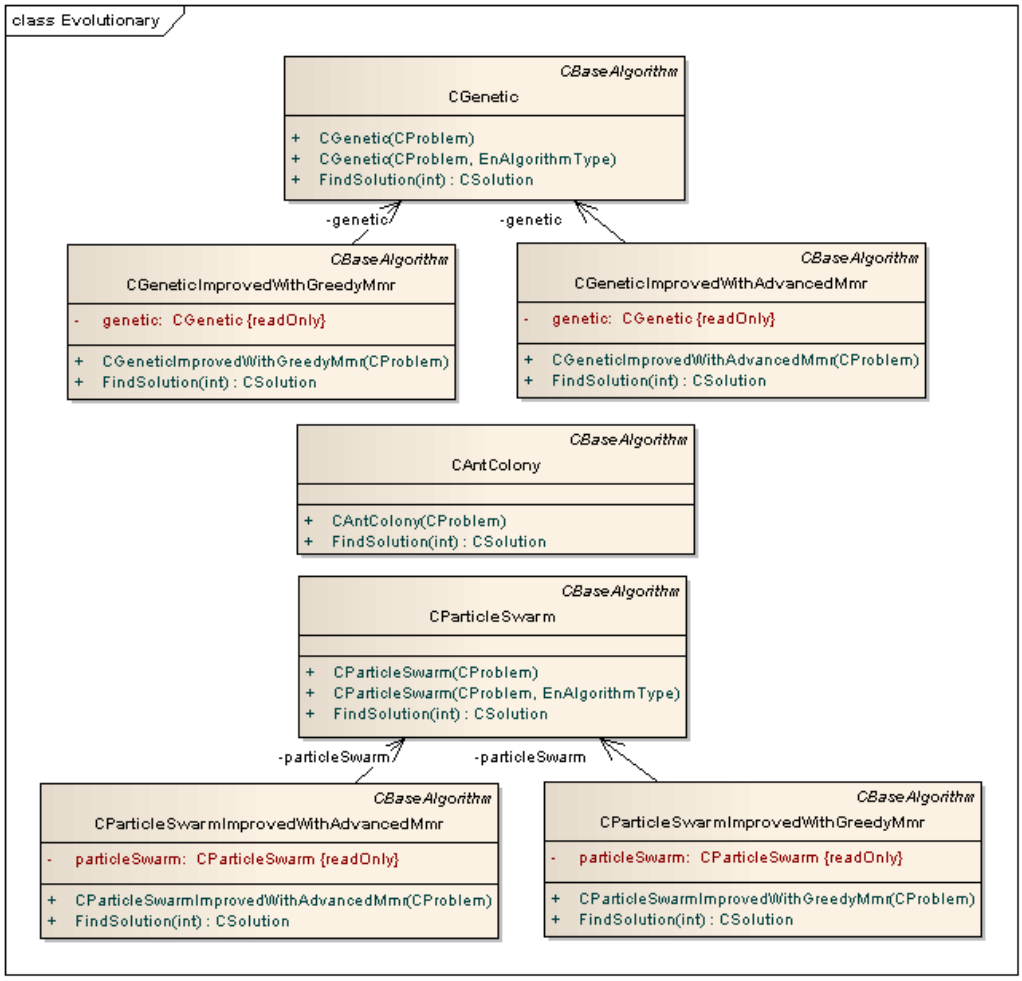

Figure 4.4: Evolutionary Algorithms Package

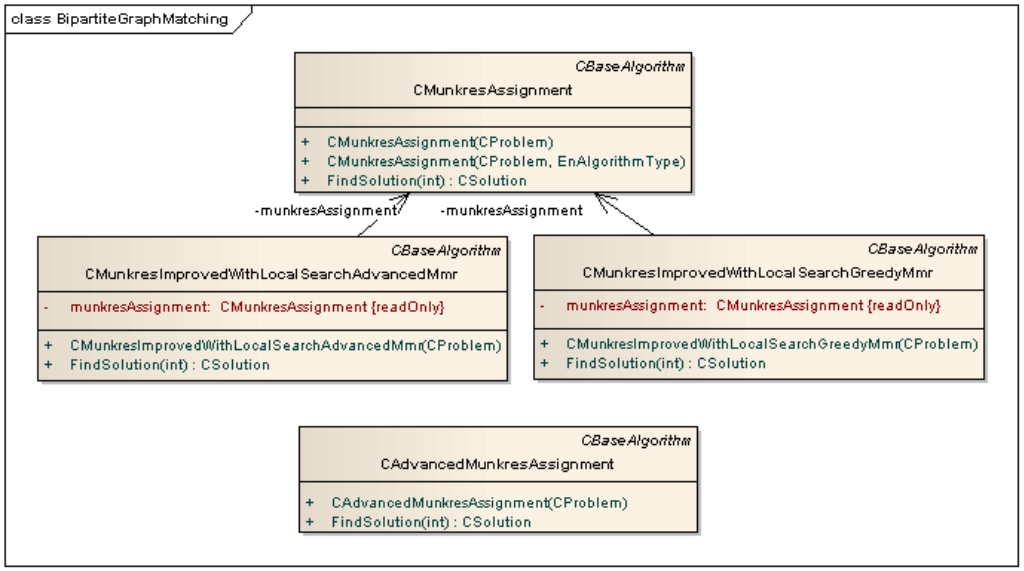

Figure 4.5: Bipartite Graph Matching Algorithms Package

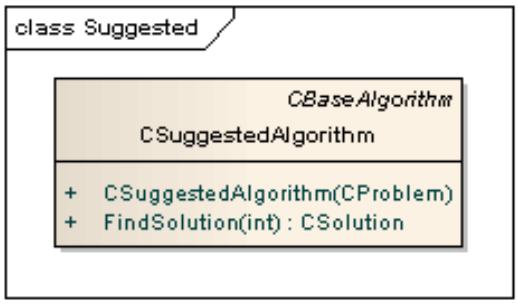

Figure 4.6: Suggested Algorithm Package

- Suggested Algorithm (Shown in Figure 4.6).

As it can be seen, all algorithms inherit from the base algorithm. If an algorithm needs to be developed, it should implement the base algorithm class.

2. **Scenario Package**

The classes of the Scenario Package is shown in Figure 4.7. This package consists of the following 5 classes:

- **Problem:** This class contains the methods for creating a problem instance (number of weapons, number of targets, target values, kill probabilities, maximum allowed time). Number of weapons, number of targets and maximum allowed time are taken from the user and the target values and kill probabilities are generated randomly by using those values.
- **Solution:** This class contains the list of allocations and the solution (objective function) value found by applying the algorithm to the problem instance.
- **Solution In Experiment:** This class contains the properties of the solution found by applying an algorithm that exists in the experiment. In addition to the list of allocations and solution value, it includes type, execution time and rank of the algorithm.
- **Experiment:** This class contains the problem instance and the list of solutions found by applying all selected algorithms that exist in the experimental setup.
- **Result:** In case of running more than one experiment at a time, this class contains the list of experiments.

3. **GUI Package**

To be able to use the testbed efficiently, a GUI is developed. The main screen of the testbed is shown in Figure 4.8.

Basically, the user enters the number of targets, the number of weapons and the maximum allowed time for running each algorithm. Then, he/she

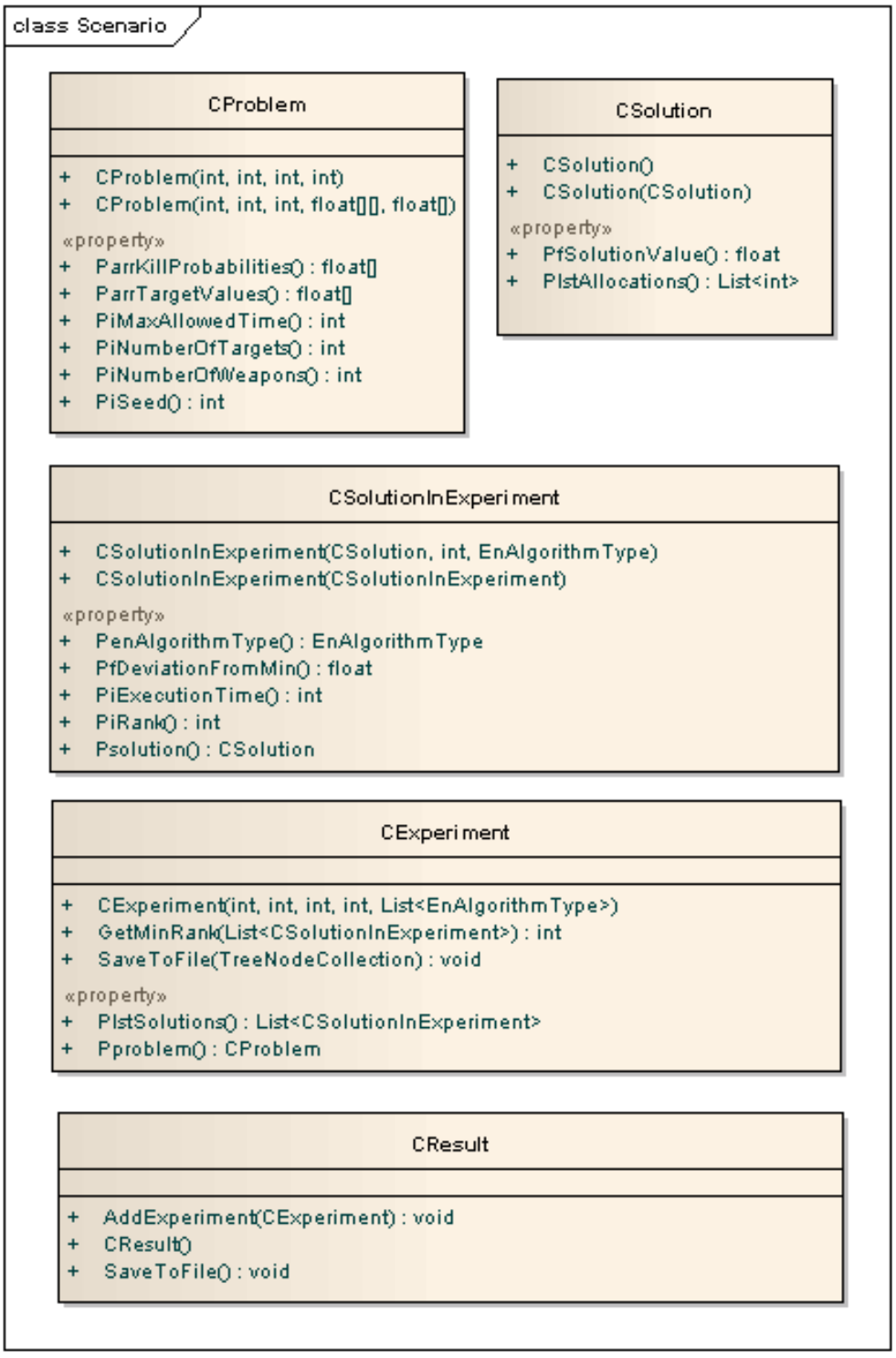

Figure 4.7: Scenario Package

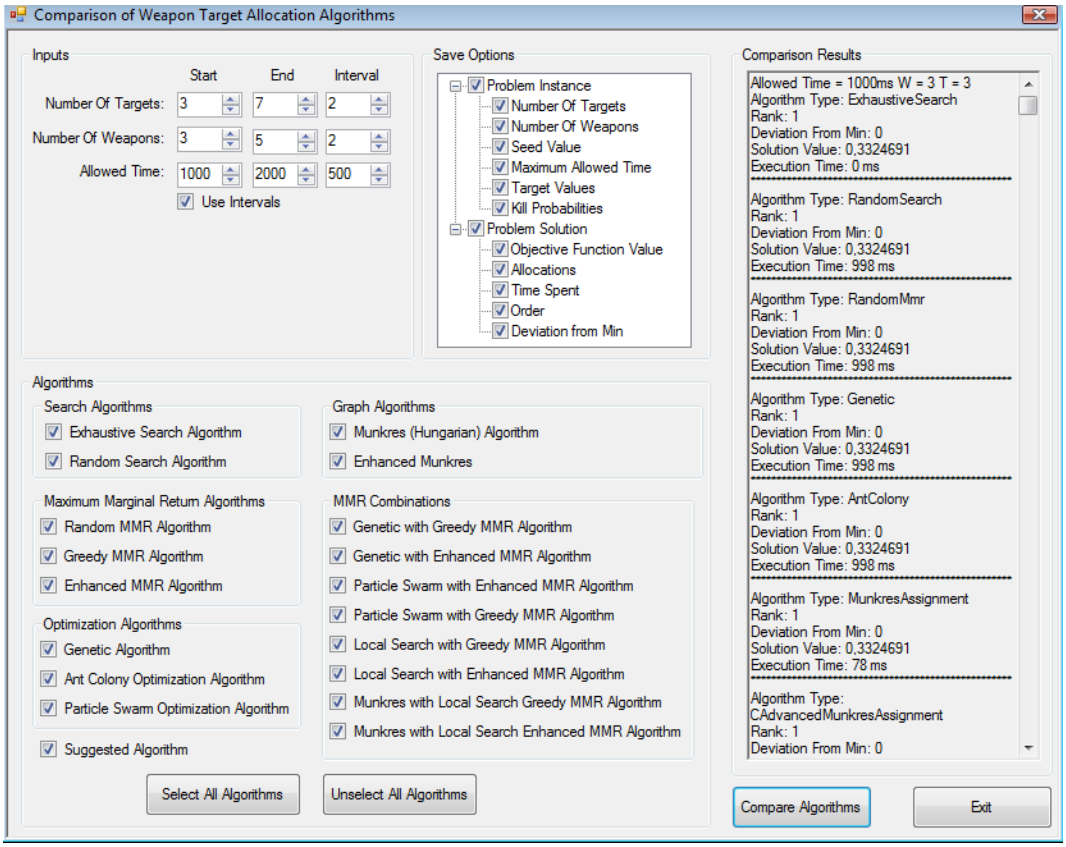

Figure 4.8: Main Screen

selects the algorithms to be compared.

If *Use Intervals* checkbox is checked, it means that more than one experiment are desired to be run. For example, for the sample screen given in Figure 4.8, 18 experiments are run with the following parameters:

- $|W| = 3$, $|T| = 3$, time = 1000 ms,
- $|W| = 3$, $|T| = 3$, time = 1500 ms,
- $|W| = 3$, $|T| = 3$, time = 2000 ms,
- $|W| = 3$, $|T| = 5$, time = 1000 ms,
- $|W| = 3$, $|T| = 5$, time = 1500 ms,
- $|W| = 3$, $|T| = 5$, time = 2000 ms,
- $|W| = 3$, $|T| = 7$, time = 1000 ms,
- $|W| = 3$, $|T| = 7$, time = 1500 ms,
- $|W| = 3$, $|T| = 7$, time = 2000 ms,
- $|W| = 5$, $|T| = 3$, time = 1000 ms,
- $|W| = 5$, $|T| = 3$, time = 1500 ms,
- $|W| = 5$, $|T| = 3$, time = 2000 ms,
- $|W| = 5$, $|T| = 5$, time = 1000 ms,
- $|W| = 5$, $|T| = 5$, time = 1500 ms,
- $|W| = 5$, $|T| = 5$, time = 2000 ms,
- $|W| = 5$, $|T| = 7$, time = 1000 ms,
- $|W| = 5$, $|T| = 7$, time = 1500 ms,
- $|W| = 5$, $|T| = 7$, time = 2000 ms.

If *Use Intervals* checkbox is not checked, a single experiment is run using the values given in *Start* column.

The comparison results are shown at the right of the screen. The results can also be saved to a file for future use. The user can pick the information by using save options.

4.2 Experimental Results

Since the evaluation of algorithms needs to be fair, all algorithms were run on the same computer. The computer has the following specifications:

- **CPU:** Intel Core2 Duo, 2.4GHz
- RAM: 3 GB
- Operation System: Microsoft XP

For the evaluation, the problem instances can be categorized into two: *large scale instances* ($10 <= |T|, |W| <= 80$) and *small scale instances* ($|T|, |W| < 10$). The minimum and maximum values are determined as 10 and 80 to be more realistic.

The implemented algorithms are applied on 40 small scale and 110 large scale problem instances. $|T|$ and $|W|$ are varied from 3 to 80 and the maximum allowed search time is varied from 100 to 2600.

4.2.1 Performance Evaluation

The performance evaluation of the implemented algorithms is done by comparing the execution times of the algorithms. The average value of maximum allowed execution time in the problem instances is 1500 ms. In Figure 4.9, the execution time of the algorithms for small scale problem instances is illustrated. As it can be seen, some of the implemented heuristic algorithms such as *Random Search*, *Random MMR*, *Genetic Algorithm*, etc. make use of the maximum allowed execution time. On the other hand, some algorithms such as Exhaustive Search, Greedy MMR, Advanced MMR, etc. executes independent of the maximum allowed execution time.

The illustration of the execution times for large scale problem instances (Figure 4.10) is similar. However, for large scale problem instances it is impossible to apply *Exhaustive Search Algorithm*, since it is exponential. So, the given illustration excludes the *Exhaustive Search Algorithm*.

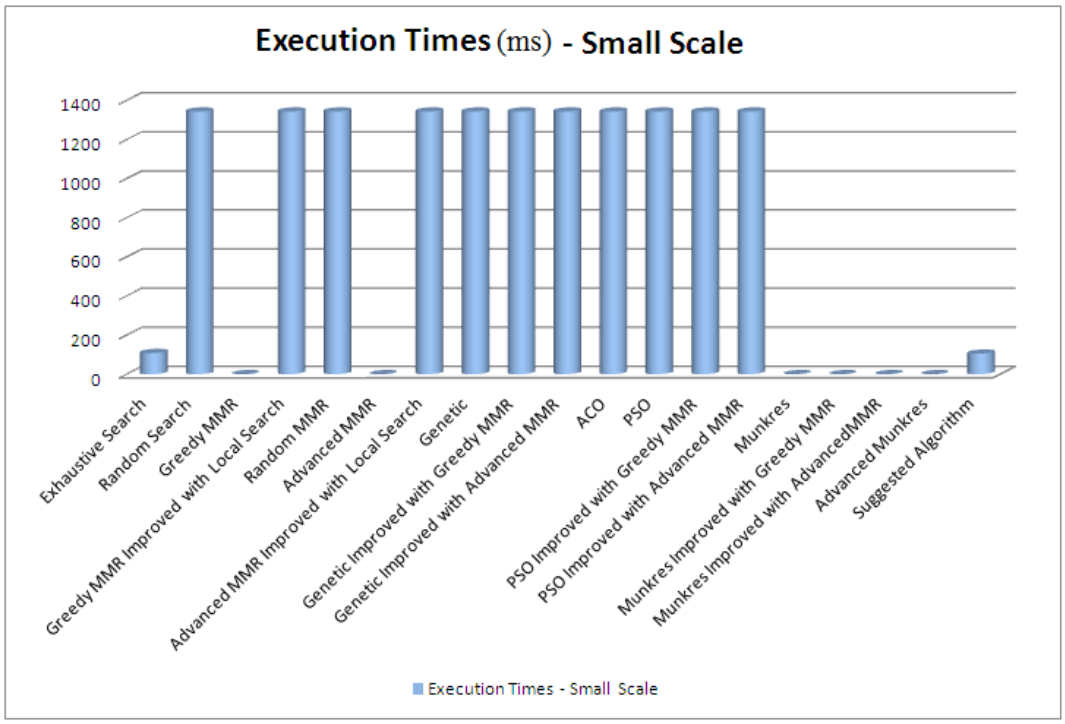

Figure 4.9: Execution Times - Small Scale Problem Instance

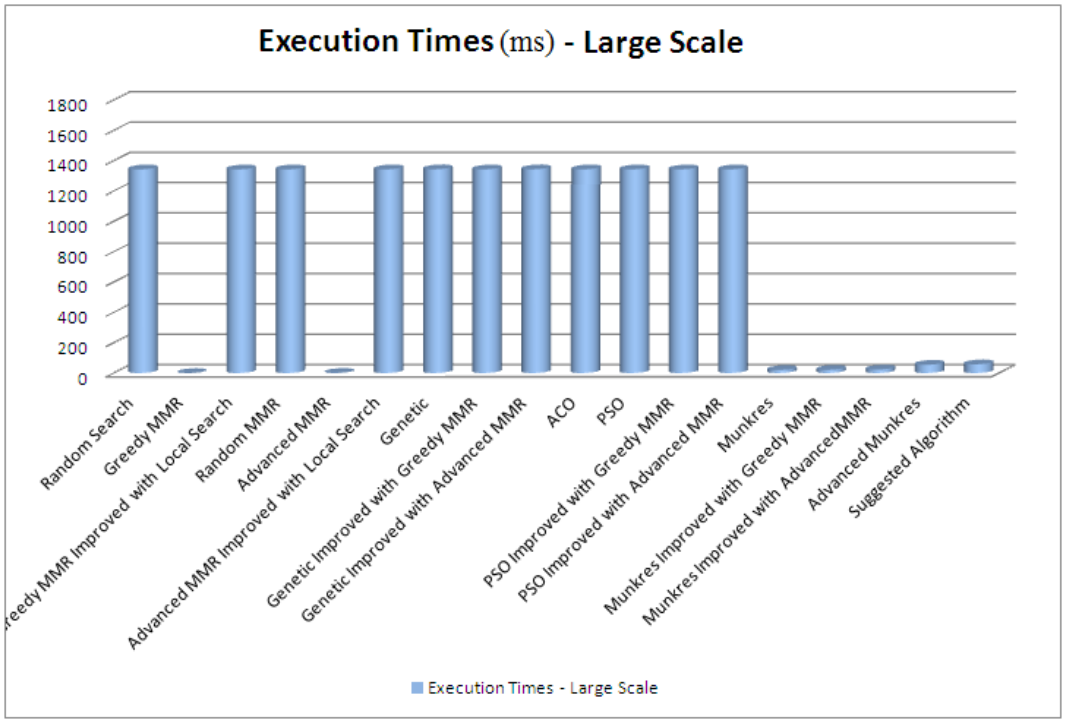

Figure 4.10: Execution Times - Large Scale Problem Instance

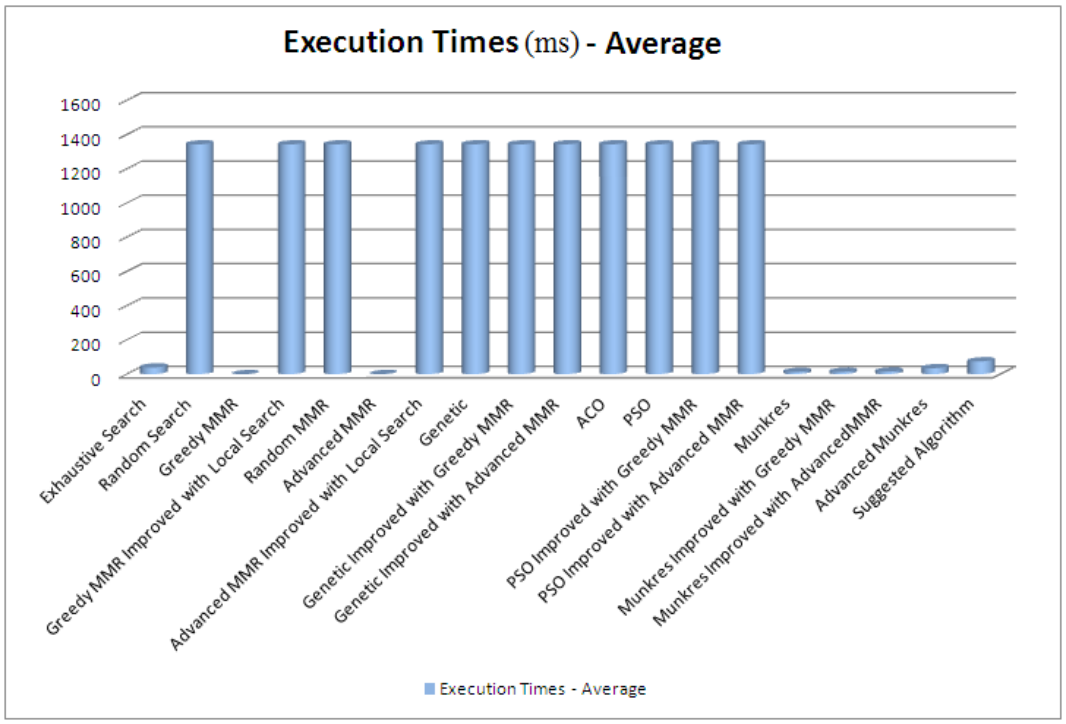

Figure 4.11: Execution Times - Average

The execution times for all problem instances is illustrated in Figure 4.11. Although there are algorithms that run with better execution times than that of the *Suggested Algorithm*, the *Suggested Algorithm* also gives a very good execution time result. Even if $|T| = |W| = 80$, the *Suggested Algorithm* returns in 46 ms.

The execution times received from applying all experiments are given in Appendix A.

4.2.2 Optimality Evaluation

As it has been stated earlier, the Weapon - Target Allocation Problem is an NP-Complete problem and almost all of the implemented algorithms are heuristic algorithms. So, it is difficult to decide whether the implemented algorithm gives the optimal solution or not. However, it is a fact that the lesser the objective function value, the nearer the solution is to the optimum.

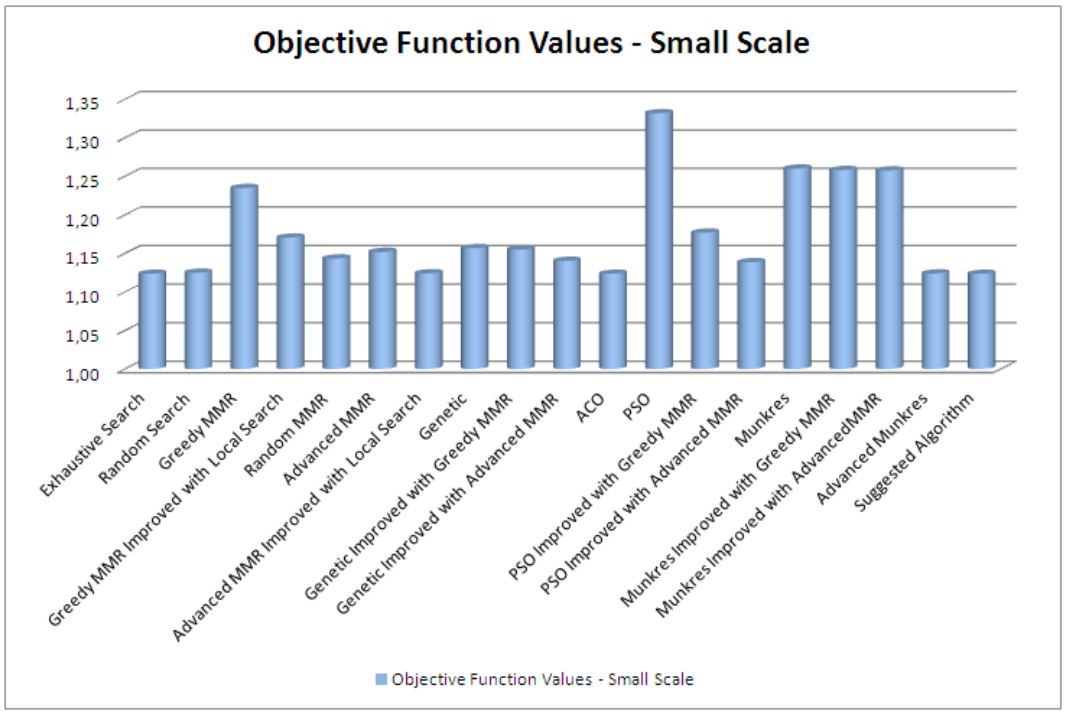

Figure 4.12: Objective Function Values - Small Scale Problem Instance

In this subsection, comparisons based on the objective function values, deviations and ranks of the algorithms are given.

4.2.2.1 Objective Function Value Comparison

In all of the experiments run, the algorithms are ordered based on the objective function values. The average objective function values for the small and large problem instances are illustrated in Figures 4.12 and 4.13.

The overall objective function values for the algorithms is illustrated in Figure 4.14. As it can be seen from the figure, the *suggested algorithm* gives the minimum objective function value.

The objective function values received from applying all experiments are given in Appendix A.

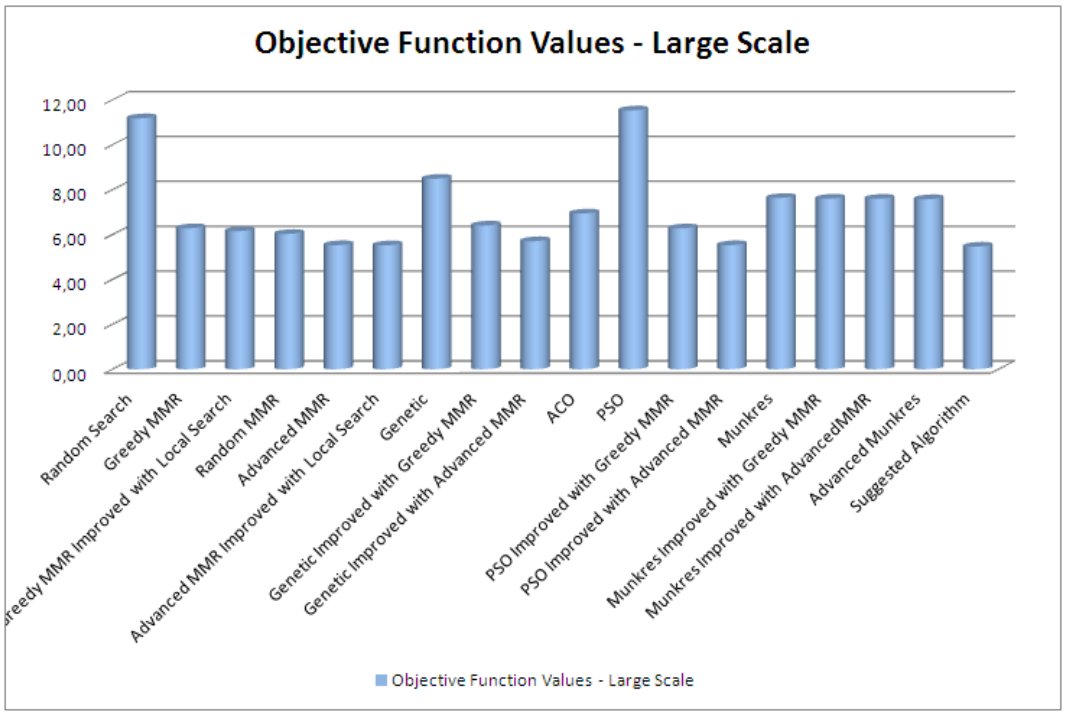

Figure 4.13: Objective Function Values - Large Scale Problem Instance

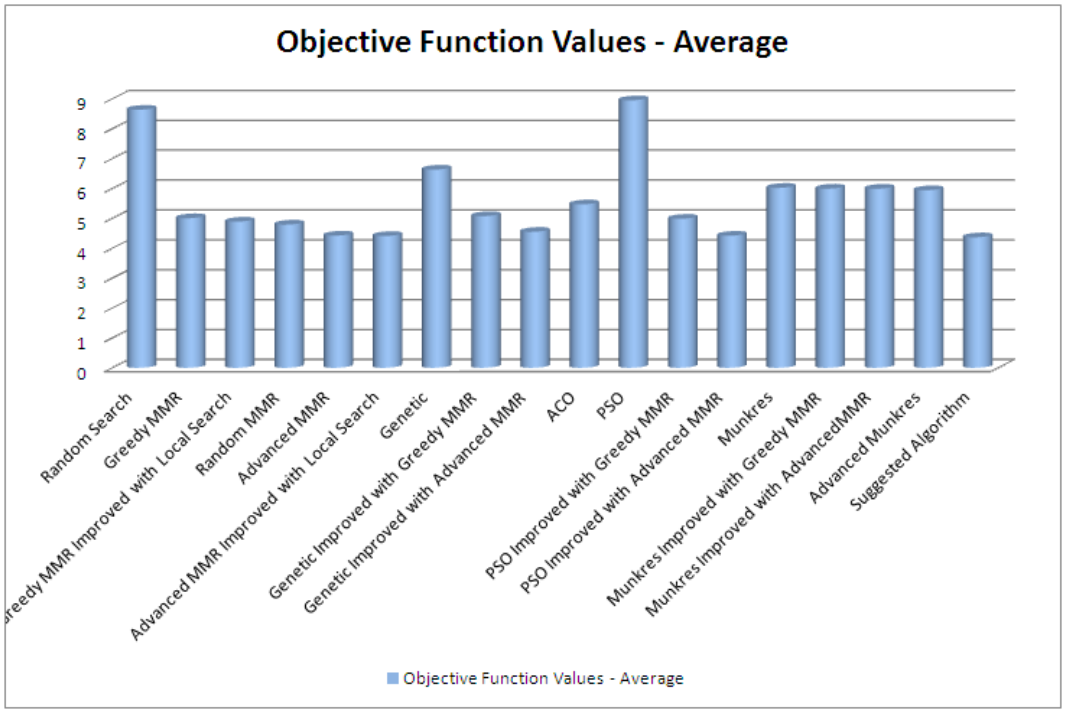

Figure 4.14: Objective Function Values - Average

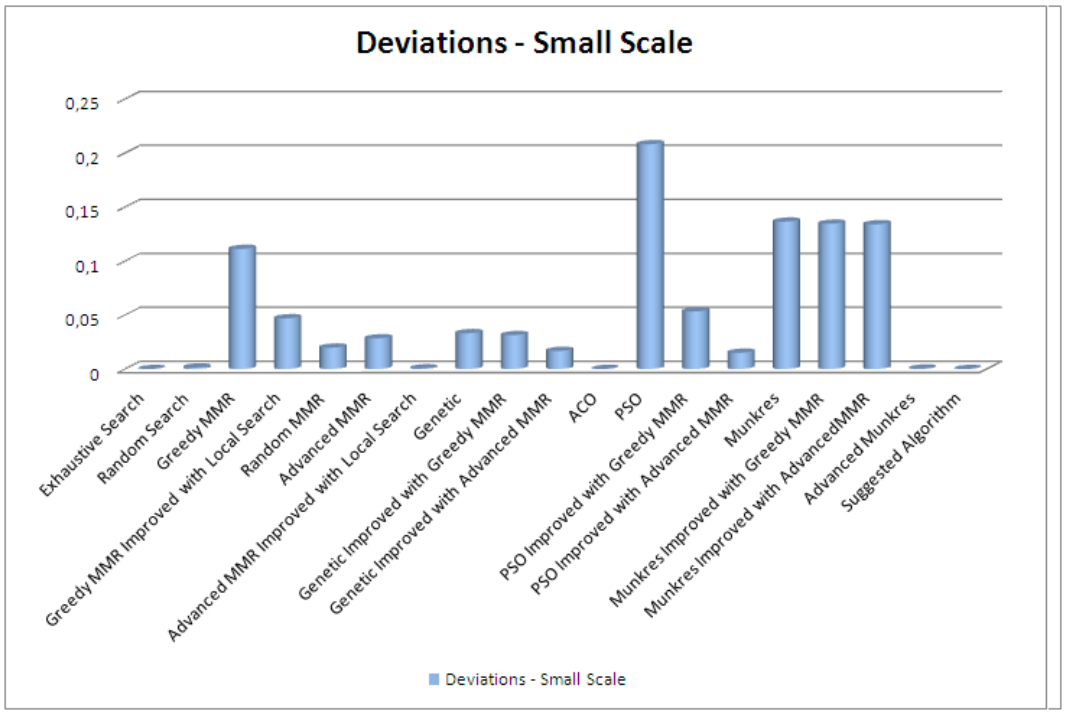

Figure 4.15: Deviations - Small Scale Problem Instance

4.2.2.2 Deviation Comparison

By using the objective function values, deviations from the minimum are plotted. The figures are similar and given in Figures 4.15, 4.16, and 4.17.

The deviations received from applying all experiments are given in Appendix A.

4.2.2.3 Rank Comparison

For each experiment conducted, a rank value is given to each algorithm based on the objective function value. If two algorithms score the same objective function value, the same rank is given to both of the algorithms. The algorithm which gives minimum objective function value is given rank 1. For small scale problem instances (illustrated in Figure 4.18), the *Exhaustive Search Algorithm* and the *Suggested Algorithm* get rank 1.

For large scale problem instances (illustrated in Figure 4.19), since the exhaus-

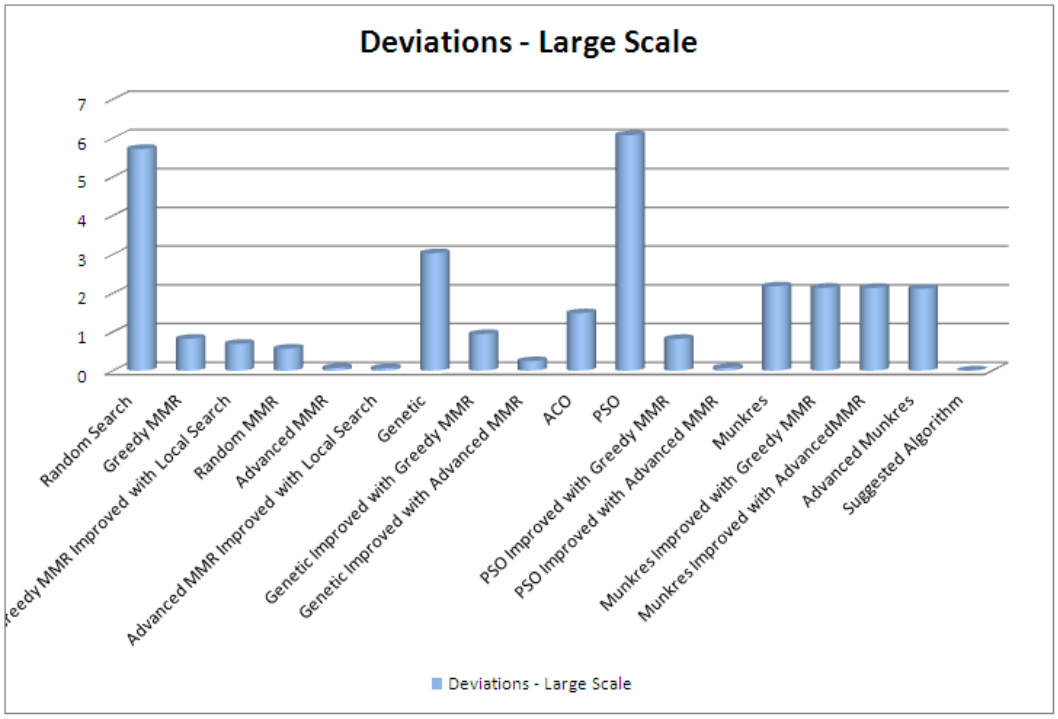

Figure 4.16: Deviations - Large Scale Problem Instance

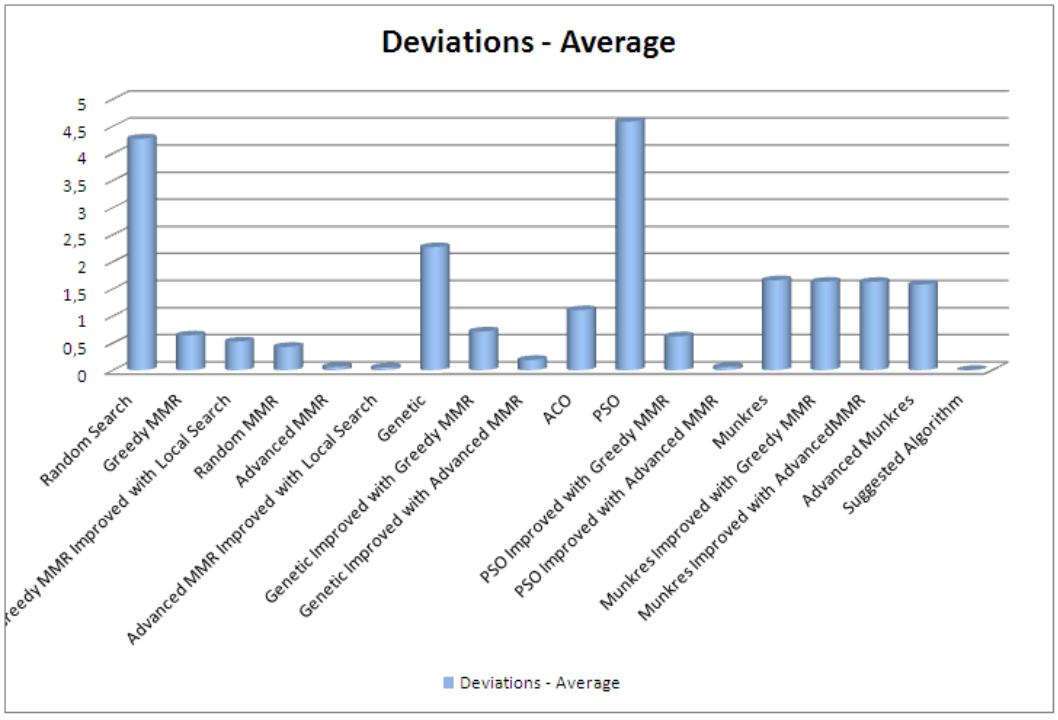

Figure 4.17: Deviations - Average

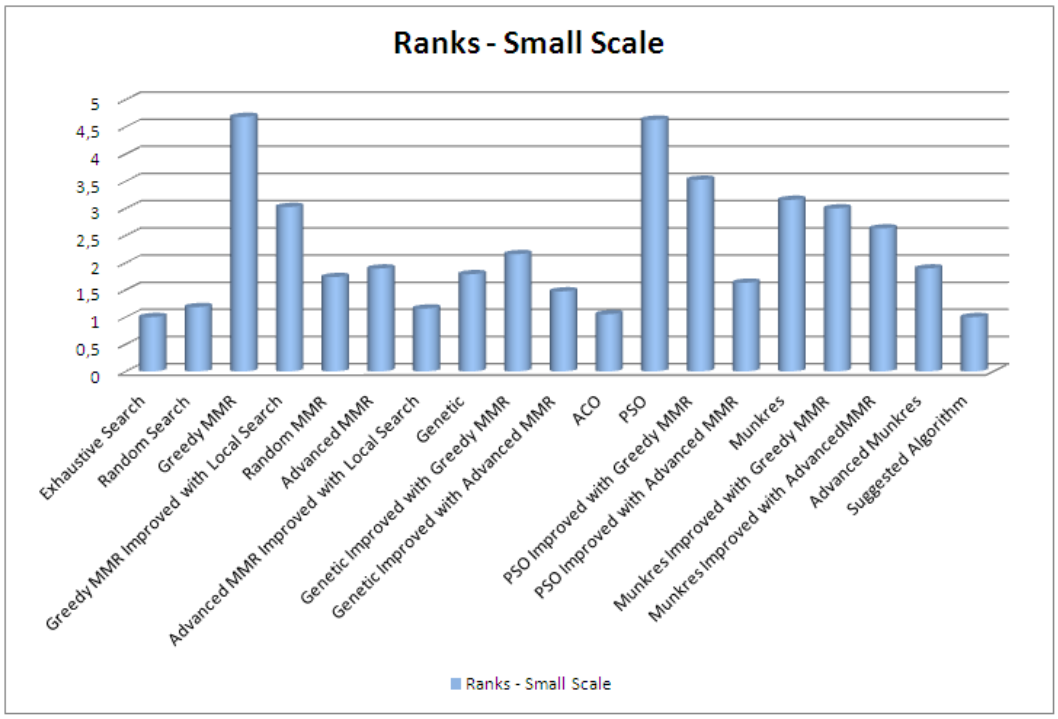

Figure 4.18: Ranks - Small Scale Problem Instance

tive search algorithm cannot be applied, it gets the worst result. However, the suggested algorithm still gets the best rank.

The average rank values for the algorithms are illustrated in Figure 4.20. So, if a comparison is made based on the optimality among the implemented algorithms, the *Suggested Algorithm* gives the result that is closest to the optimal solution.

Also, it can be interpreted from the figures that the improved algorithms give better results than the original algorithms. For example, *Genetic Algorithm Improved with MMR Algorithms* give better result than the *Genetic Algorithm* and *Particle Swarm Optimization Algorithm Improved with MMR Algorithms* give better result than the *Particle Swarm Optimization Algorithm*, etc.

The ranks are compared based on the algorithm categories (*Search Algorithms*, *MMR Algorithms*, *Evolutionary Algorithms* and *Suggested Algorithm*) in Figure 4.21. Although the implementation of the MMR algorithms is easier, they produce very good results. Bipartite graph matching algorithms give also good results.

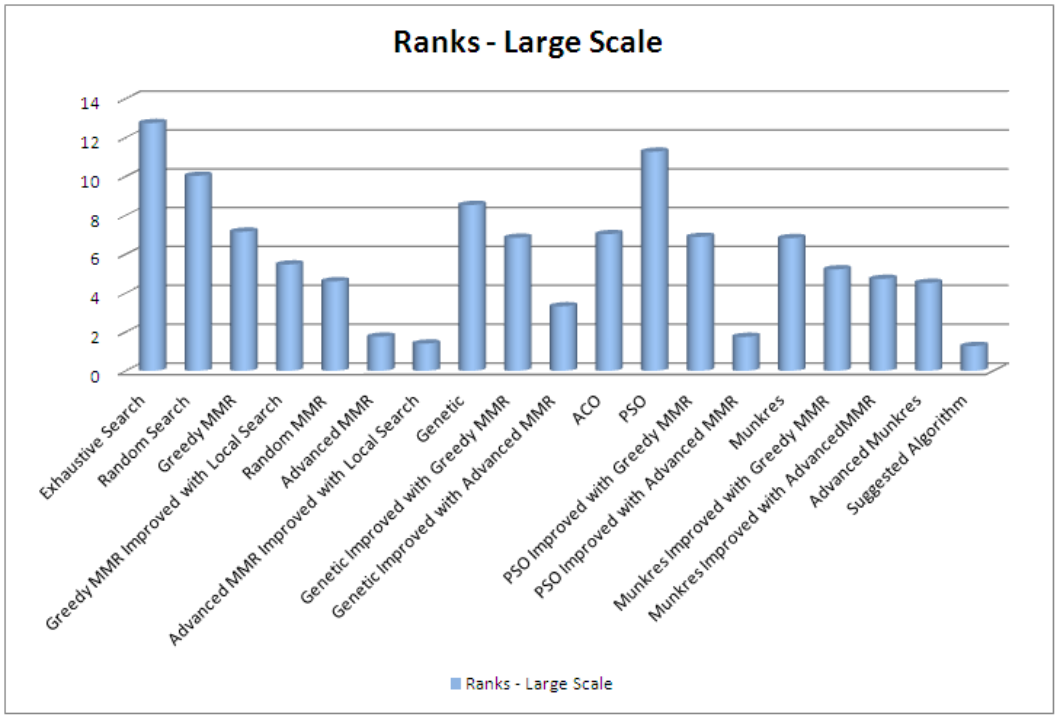

Figure 4.19: Ranks - Large Scale Problem Instance

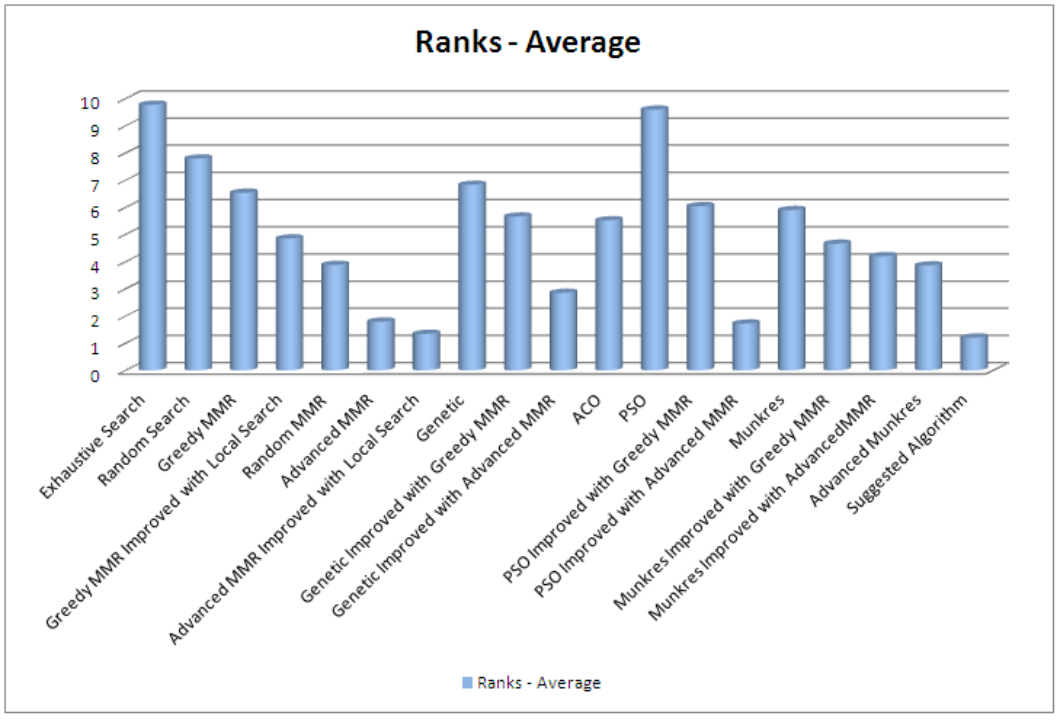

Figure 4.20: Ranks - Average

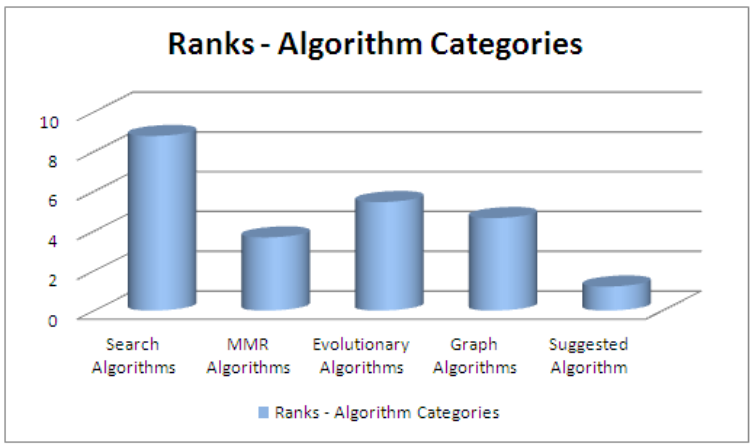

Figure 4.21: Ranks - Algorithm Categories

The rank values received from applying all experiments are given in Appendix A.

CHAPTER 5

CONCLUSION AND FUTURE WORK

This thesis study presents an efficient algorithm for static target-based Weapon - Target Allocation problem that can be used in real ground-based air defence systems. In this thesis, a mathematical formulation of the Weapon - Target Allocation problem is given. Since the Weapon - Target Allocation problem is an NP-Complete problem, there is no exact algorithm that gives the optimal solution in polynomial time. So, heuristic algorithms are developed to solve this problem. The problem is considered with respect to other known problems and the algorithms for those problems are used in the solution. Search algorithms, MMR algorithms, evolutionary algorithm and bipartite graph matching algorithms are implemented in order to solve the problem. The algorithms are improved by making use of the maximum allowed time criterion. A simulation environment is developed to be able to compare the algorithms and to allow users to test new algorithms easily. An algorithm is proposed by making use of the algorithms that give better results. This algorithm is compared to the other algorithms and it is seen that the suggested algorithm gives the best (closest to the optimal solution) result among all. The comparisons of the algorithms for the Weapon - Target Allocation problem in terms of optimality, performance and efficiency criteria are presented.

For all algorithms that are implemented, the pseudocodes are given to be more clear. For testing the performance and correctness of algorithms, the developed simulation environment is used. To be more understandable, the results are illustrated graphically. Also, the details of the computational results of the

comparisons are given in Appendix A.

In this thesis, the target values which are the output of the Threat Evaluation phase are generated randomly and supplied directly to the algorithms. This thesis work can be extended to deal with the whole TEWA system, including the Threat Evaluation phase. Also, the kill probabilities are generated randomly. The reason for generating these values randomly is that the determination of these values requires high domain knowledge (such as the knowledge of all possible target types and the properties of targets, existing weapon types and the properties of weapons, etc.). To be more realistic, methods for determining these values in real-time can be developed with a domain expert as a future work.

In this thesis, only the static version of the Weapon - Target Allocation problem is studied. In the static version, all weapons are engaged to targets in a single stage. Although there are cases in which the assignment and firing of weapons are made simultaneously, usually it is not the case. The outcomes of the engagements may affect the next engagement stage in actual cases. Yet, the algorithms implemented within this thesis work can be applied periodically to deal with the dynamic environment. However, there may be fluctuations in the assignments due to the application of the algorithm periodically to the problem. To solve fluctuation problem, some fixing rules may be applied. The dynamic version of the Weapon - Target Allocation problem can be studied as a future work. The simulation environment that is developed for the comparison of algorithms can be used for comparing the dynamic versions of algorithms. This thesis work actually forms a basis for the study of the dynamic version.

By making use of the simulation environment, more algorithms can be implemented and compared to existing algorithms easily.

REFERENCES

[1] S. Lloyd, H. Witsenhausen, "Weapon Allocation is NP-complete", *Proceedings of the 1986 Summer Conference on Simulation*, 1986.

[2] NATO Standardization Agency, "NATO Glossary of Terms and Definitions", *AAP-6*, 2008.

[3] S. Paradis, A. Benaskeur, M. Oxenham and P. Cutler, "Threat Evaluation and Weapon Allocation in Network-Centric Warfare", *7th International Conference on Information Fusion*, pp. 1078–1085, 2005.

[4] K. P. Werrell, "A Short Operational History of Ground-Based Air Defense", Air University Press *BELLSYS*, vol. 27, pp. 379–423, 1948.

[5] F. Joohansson, G. Falkman, "A Bayesian Network Approach to Threat Evaluation with Application to an Air Defense Scenario", *11th International Conference on Information Fusion*, pp. 1352–1358.

[6] M. Liebhaber, B. Feher, "Air Threat Assessment: Research, Model, and Display Guidelines", *Command and Control Research and Technology Symposium*, 2002.

[7] M. Liebhaber, B. Feher, "Surface Warfare Threat Assessment: Requirements Definition", *Technical Report 1887*, SSC San Diego, 2002.

[8] J. N. Roux, J. H. Van Vuuren, "Real Time Threat Evaluation in a Ground Based Air Defence Environment", *ORiON*, vol. 24, pp. 75–100.

[9] Y. Liang, "An Approximate Reasoning Model for Situation and Threat Assessment", *4th International Conference on Fuzzy Systems and Knowledge Discovery*, 2007.

[10] N. Okello, G. Thoms, "Threat Assessment Using Bayesian Networks", *6th International Conference on Information Fusion*, 2003.

[11] F. Johansson, G. Falkman, "A Comparison Between Two Approaches to Threat Evaluation in an Air Defense Scenario", *5th International Conference on Modeling Decisions for Artificial Intelligence*, 2008.

[12] R. K. Ahuja, A. Kumar, K. C. Jha, J. B. Orlin, "Exact and Heuristic Algorithms for the Weapon-Target Assignment Problem", *Massachusetts Institute of Technology Press*, Cambridge, 2007.

[13] P. A. Hosein, M. Athans, "Some Analytical Results for the Dynamic Weapon-Target Allocation Problem", *Massachusetts Institute of Technology Press*, Cambridge, 1990.

[14] S. P. Lloyd, H. S. Witsenhausen, "Weapons Allocation is NP-Complete", *Summer Conference on Simulation*, Reno, 1986.

[15] J. M. Rosenberger, H. S. Hwang, R. P. Pallerla, A. Yücel, R. L. Wilson, E. G. Brungardt, "The Generalized Weapon Target Assignment Problem", *10th International Command and Control Research and Technology Symposium*, 2005.

[16] M. Chan, "A Genetic Algorithm for the Weapon to Target Assignment Problem".

[17] A. Tebo, "Sensor Fusion Employs a Variety of Architecture, Algorithms, and Applications", *OE Reports (the International Society for Optical Engineering)*, 1997.

[18] B. Z. Zabinsky, "Random Search Algorithms", *University of Washington, Seattle, WA*, 2009.

[19] P. A. Hosein, J. T. Walton, M. Athans, "Dynamic Weapon-Target Assignment Problems with Vulnerable C^2 Nodes", *Massachusetts Institute of Technology Press*, Laboratory for Information and Decision Systems, 1988.

[20] H. Pohlheim, "GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with MATLAB Documentation", 2006.

[21] M. Dorigo, M. Birattari, T. Stützle, "Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique", Universite Libre de Bruxelles, Belgium, 2006.

[22] F. Johansson, "Evaluating the Performance of TEWA Systems", PhD Thesis, University of Skövde, 2010.

[23] H. A. Baier Saip, C. L. Lucchesi, "Matching Algorithms for Bipartite Graphs", Universidade Estadual de Campinas, Brasil, 1993.

APPENDIX A

COMPUTATIONAL RESULTS OF THE EXPERIMENTS

The execution times, objective function values, deviations and ranks of the algorithms are given in the following tables. The numbers for the algorithm names are given below:

1. Exhaustive Search Algorithm
2. Random Search Algorithm
3. Greedy MMR Algorithm
4. Greedy MMR Algorithm Improved with Local Search
5. Random MMR Algorithm
6. Advanced MMR Algorithm
7. Advanced MMR Algorithm Improved with Local Search
8. Genetic Algorithm
9. Genetic Algorithm Improved with Greedy MMR Algorithm
10. Genetic Algorithm Improved with Advanced MMR Algorithm
11. Ant Colony Optimization Algorithm
12. Particle Swarm Optimization Algorithm
13. Particle Swarm Optimization Algorithm Improved with Greedy MMR Algorithm

14. Particle Swarm Optimization Algorithm Improved with Advanced MMR Algorithm
15. Munkres' Assignment Algorithm
16. Munkres' Assignment Algorithm Improved with Greedy MMR Algorithm
17. Munkres' Assignment Algorithm Improved with Advanced MMR Algorithm
18. Advanced Munkres' Assignment Algorithm
19. Suggested Algorithm

Table A.1: Execution Times (in ms) - 1

T	W	Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	20	100	0	93	0	93	93	0	93	93	93	93	93	93	93	93	0	0	0	0	0
40	20	100	0	93	0	93	93	0	93	93	93	93	109	93	93	93	15	0	0	0	0
60	20	100	0	93	0	93	93	0	93	93	93	93	171	93	93	93	0	0	0	15	0
80	20	100	0	93	0	93	93	0	93	93	93	93	156	93	93	93	15	0	15	0	15
20	40	100	0	93	0	93	93	0	93	93	93	93	125	93	93	93	0	0	0	15	15
40	40	100	0	93	0	93	93	0	93	93	93	93	125	93	93	93	0	0	0	0	0
60	40	100	0	93	0	93	93	0	93	93	93	93	281	93	93	93	15	0	15	0	15
80	40	100	0	93	0	93	93	0	93	93	93	93	500	93	93	93	31	15	31	15	15
20	60	100	0	93	0	93	93	0	93	93	93	93	187	93	93	93	31	15	31	78	78
40	60	100	0	93	0	93	93	0	93	93	93	93	390	93	93	93	15	15	15	46	46
60	60	100	0	93	0	93	93	0	93	93	93	93	593	93	93	93	15	15	0	0	15
80	60	100	0	93	0	93	93	0	93	125	93	93	1062	93	93	93	31	31	31	31	46
20	80	100	0	93	0	93	93	0	93	93	93	93	437	93	93	93	62	109	109	281	281
40	80	100	0	93	0	93	93	0	93	93	93	93	890	93	93	93	46	78	46	171	171
60	80	100	0	93	0	93	93	0	93	93	93	93	1343	93	93	93	31	46	46	140	140
80	80	100	0	93	0	93	93	15	93	93	93	109	1765	93	93	93	46	31	46	31	46
20	20	600	0	593	0	593	593	0	593	593	593	593	593	593	593	593	0	0	0	0	0
40	20	600	0	593	0	593	593	0	593	593	593	593	609	593	593	593	0	0	0	0	0
60	20	600	0	593	0	593	593	0	593	593	593	593	625	593	593	593	15	0	0	0	0
80	20	600	0	593	0	593	593	0	593	593	593	593	640	593	593	593	15	0	15	15	15
20	40	600	0	593	0	593	593	0	593	593	593	593	640	593	593	593	0	0	0	15	15
40	40	600	0	593	0	593	593	0	593	593	593	593	625	593	593	593	15	0	0	0	0
60	40	600	0	593	0	593	593	0	593	593	593	593	859	593	593	593	15	15	0	0	15
80	40	600	0	593	0	593	593	0	593	593	593	593	1015	593	593	593	31	15	31	15	31
20	60	600	0	593	0	593	593	0	593	593	593	593	781	593	593	593	15	15	31	93	62
40	60	600	0	593	0	593	593	0	593	593	593	593	781	593	593	593	15	15	15	46	46
60	60	600	0	593	0	593	593	0	593	593	593	593	593	593	593	593	15	0	15	0	15
80	60	600	0	593	0	593	593	0	593	593	593	593	1062	593	593	593	31	31	31	31	46
20	80	600	0	593	0	593	593	0	593	593	593	593	890	593	593	593	62	109	109	281	281
40	80	600	0	593	0	593	593	0	593	593	593	593	890	593	593	593	46	46	46	203	203
60	80	600	0	593	0	593	593	0	593	593	593	593	1328	593	593	593	46	46	46	156	140
80	80	600	0	593	0	593	593	15	593	593	593	593	1796	593	593	593	31	31	31	46	46
20	20	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1093	1093	1093	1093	0	0	0	0	0
40	20	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1093	1093	1093	1093	0	0	0	0	0
60	20	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1093	1093	1093	1093	0	0	0	0	0
80	20	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1140	1093	1093	1093	0	0	15	15	15
20	40	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1140	1093	1093	1093	15	0	0	0	15
40	40	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1140	1093	1093	1093	0	0	0	0	15

Table A.2: Execution Times (in ms) - Cont'd

T	W	Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
60	40	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1140	1093	1093	1093	15	15	0	0	15
80	40	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1531	1093	1093	1093	15	15	31	31	15
20	60	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1171	1093	1093	1093	31	15	15	78	78
40	60	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1171	1093	1093	1093	15	15	15	31	46
60	60	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1171	1093	1093	1093	15	0	15	15	15
80	60	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	2093	1093	1093	1093	46	31	31	31	46
20	80	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1328	1093	1093	1093	62	109	109	281	281
40	80	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1781	1093	1093	1093	31	78	46	171	171
60	80	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1328	1093	1093	1093	31	46	46	140	140
80	80	1100	0	1093	0	1093	1093	15	1093	1109	1093	1093	1781	1093	1093	1093	31	31	31	31	62
20	20	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1593	1593	1593	1593	0	0	0	0	0
40	20	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1625	1593	1593	1593	0	0	0	0	0
60	20	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1656	1593	1593	1593	0	0	0	0	15
80	20	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1609	1593	1593	1593	15	0	15	15	15
20	40	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1593	1593	1593	1593	0	0	0	15	15
40	40	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1656	1593	1593	1593	0	0	0	0	15
60	40	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1718	1593	1593	1593	15	0	15	0	15
80	40	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	2046	1593	1593	1593	15	15	31	31	15
20	60	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1765	1593	1593	1593	15	15	15	93	93
40	60	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1953	1593	1593	1593	15	15	15	46	46
60	60	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1765	1593	1593	1593	0	0	15	15	15
80	60	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	2093	1593	1593	1593	46	31	31	31	46
20	80	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1765	1593	1593	1593	62	109	109	312	281
40	80	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1781	1593	1593	1593	46	78	46	171	171
60	80	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	2703	1593	1593	1593	46	46	46	156	140
80	80	1600	0	1593	0	1593	1593	15	1593	1593	1593	1609	1765	1593	1593	1593	31	31	31	46	46
20	20	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2093	2093	2093	2093	0	0	0	0	0
40	20	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2109	2093	2093	2093	0	0	0	0	0
60	20	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2171	2093	2093	2093	15	0	0	0	0
80	20	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2187	2093	2093	2093	15	0	15	0	15
20	40	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2109	2093	2093	2093	0	0	0	15	15
40	40	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2156	2093	2093	2093	15	0	0	0	15
60	40	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2281	2093	2093	2093	15	0	15	15	15
80	40	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2546	2093	2093	2093	15	15	31	31	31
20	60	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2156	2093	2093	2093	31	15	15	78	93
40	60	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2343	2093	2093	2093	15	15	15	46	46
60	60	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2343	2093	2093	2093	15	0	15	15	15
80	60	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2109	2093	2093	2093	31	31	31	46	46

Table A.3: Execution Times (in ms) - Cont'd

T	W	Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	80	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2218	2093	2093	2093	62	109	109	281	296
40	80	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2656	2093	2093	2093	46	78	46	171	171
60	80	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2656	2093	2093	2093	46	46	46	156	140
80	80	2100	0	2093	0	2093	2093	15	2093	2109	2093	2093	3546	2093	2093	2093	31	31	31	46	62
20	20	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2593	2593	2593	2593	0	0	0	0	0
40	20	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2593	2593	2593	2593	15	0	0	0	0
60	20	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2640	2593	2593	2593	0	0	0	15	0
80	20	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2593	2593	2593	2593	15	0	15	0	15
20	40	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2640	2593	2593	2593	0	0	0	15	15
40	40	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2687	2593	2593	2593	0	0	0	0	15
60	40	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2875	2593	2593	2593	0	15	0	15	15
80	40	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	3062	2593	2593	2593	15	15	31	31	31
20	60	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2750	2593	2593	2593	15	15	15	93	93
40	60	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2734	2593	2593	2593	15	15	15	46	46
60	60	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2937	2593	2593	2593	15	0	15	0	15
80	60	2600	0	2593	0	2593	2593	0	2593	2593	2593	2625	3203	2593	2593	2593	31	31	31	31	46
20	80	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2656	2593	2593	2593	62	109	109	281	281
40	80	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2656	2593	2593	2593	46	78	46	140	171
60	80	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2671	2593	2593	2593	31	46	46	156	171
80	80	2600	0	2593	0	2593	2593	15	2593	2593	2593	2593	3546	2593	2593	2593	31	31	46	46	46
3	3	100	0	93	0	93	93	0	93	93	93	93	93	93	93	93	0	0	0	0	0
6	3	100	0	93	0	93	93	0	93	93	93	93	93	93	93	93	0	0	0	0	0
9	3	100	0	93	0	93	93	0	93	93	93	93	93	93	93	93	0	0	0	0	0
3	6	100	0	93	0	93	93	0	93	93	93	93	93	93	93	93	0	0	0	0	0
6	6	100	0	93	0	93	93	0	93	93	93	93	93	93	93	93	0	0	0	0	0
9	6	100	0	93	0	93	93	0	93	93	93	93	93	93	93	93	0	0	0	0	0
3	9	100	46	93	0	93	93	0	93	93	93	93	93	93	93	93	0	0	0	0	31
6	9	100	0	93	0	93	93	0	93	93	93	93	93	93	93	93	0	0	0	0	0
9	9	100	0	93	0	93	93	0	93	93	93	93	93	93	93	93	0	0	0	0	0
3	3	600	0	593	0	593	593	0	593	593	593	593	593	593	593	593	0	0	0	0	0
6	3	600	0	593	0	593	593	0	593	593	593	593	593	593	593	593	0	0	0	0	0
9	3	600	0	593	0	593	593	0	593	593	593	593	593	593	593	593	0	0	0	0	0
3	6	600	0	593	0	593	593	0	593	593	593	593	593	593	593	593	0	0	0	0	0
6	6	600	109	593	0	593	593	0	593	593	593	593	593	593	593	593	0	0	0	0	93
9	6	600	0	593	0	593	593	0	593	593	593	593	593	593	593	593	0	0	0	0	0
3	9	600	46	593	0	593	593	0	593	593	593	593	593	593	593	593	0	0	0	0	31
6	9	600	0	593	0	593	593	0	593	593	593	593	593	593	593	593	0	0	0	0	0
9	9	600	0	593	0	593	593	0	593	593	593	593	593	593	593	593	0	0	0	0	0

Table A.4: Execution Times (in ms) - Cont'd

T	W	Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
3	3	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1093	1093	1093	1093	0	0	0	0	0
6	3	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1093	1093	1093	1093	0	0	0	0	0
9	3	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1093	1093	1093	1093	0	0	0	0	0
3	6	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1093	1093	1093	1093	0	0	0	0	0
6	6	1100	109	1093	0	1093	1093	0	1093	1093	1093	1093	1093	1093	1093	1093	0	0	0	0	109
9	6	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1093	1093	1093	1093	0	0	0	0	0
3	9	1100	46	1093	0	1093	1093	0	1093	1093	1093	1093	1093	1093	1093	1093	0	0	0	0	31
6	9	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1093	1093	1093	1093	0	0	0	0	0
9	9	1100	0	1093	0	1093	1093	0	1093	1093	1093	1093	1093	1093	1093	1093	0	0	0	0	0
3	3	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1593	1593	1593	1593	0	0	0	0	0
6	3	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1593	1593	1593	1593	0	0	0	0	0
9	3	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1593	1593	1593	1593	0	0	0	0	0
3	6	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1593	1593	1593	1593	0	0	0	0	0
6	6	1600	125	1593	0	1593	1593	0	1593	1593	1593	1593	1593	1593	1593	1593	0	0	0	0	109
9	6	1600	1687	1593	0	1593	1593	0	1593	1593	1593	1593	1593	1593	1593	1593	0	0	0	0	1656
3	9	1600	31	1593	0	1593	1593	0	1593	1593	1593	1593	1593	1593	1593	1593	0	0	0	0	31
6	9	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1593	1593	1593	1593	0	0	0	0	0
9	9	1600	0	1593	0	1593	1593	0	1593	1593	1593	1593	1593	1593	1593	1593	0	0	0	0	0
3	3	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2093	2093	2093	2093	0	0	0	0	0
6	3	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2093	2093	2093	2093	0	0	0	0	0
9	3	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2093	2093	2093	2093	0	0	0	0	0
3	6	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2093	2093	2093	2093	0	0	0	0	0
6	6	2100	109	2093	0	2093	2093	0	2093	2093	2093	2093	2093	2093	2093	2093	0	0	0	0	109
9	6	2100	1671	2093	0	2093	2093	0	2093	2093	2093	2093	2093	2093	2093	2093	0	0	0	0	1640
3	9	2100	46	2093	0	2093	2093	0	2093	2093	2093	2093	2093	2093	2093	2093	0	0	0	0	31
6	9	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2093	2093	2093	2093	0	0	0	0	0
9	9	2100	0	2093	0	2093	2093	0	2093	2093	2093	2093	2093	2093	2093	2093	0	0	0	0	0
3	3	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2593	2593	2593	2593	0	0	0	0	0
6	3	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2593	2593	2593	2593	0	0	0	0	0
9	3	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2593	2593	2593	2593	0	0	0	0	0
3	6	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2593	2593	2593	2593	0	0	0	0	0
6	6	2600	109	2593	0	2593	2593	0	2593	2593	2593	2593	2593	2593	2593	2593	0	0	0	0	109
9	6	2600	1656	2593	0	2593	2593	0	2593	2593	2593	2593	2593	2593	2593	2593	0	0	0	0	1640
3	9	2600	46	2593	0	2593	2593	0	2593	2593	2593	2593	2593	2593	2593	2593	0	0	0	0	31
6	9	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2593	2593	2593	2593	0	0	0	0	0
9	9	2600	0	2593	0	2593	2593	0	2593	2593	2593	2593	2593	2593	2593	2593	0	0	0	0	0
AVG:		1350	39	1343	**0**	1343	1343	1	1343	1343	1343	1343	1515	1343	1343	1343	13	14	15	35	75

Table A.5: Objective Function Values - 1

T	W	Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	20	100	INF	3,5	1,5	1,5	1,3	0,6	0,6	2,4	1,9	0,7	1,5	3,5	1,5	0,6	0,4	0,4	0,4	0,4	0,4
40	20	100	INF	14,2	10,0	9,9	10,1	9,5	9,5	12,8	10,2	10,1	11,0	14,6	10,0	9,5	13,4	13,4	13,4	13,4	9,5
60	20	100	INF	26,6	21,3	21,3	21,3	21,0	21,0	24,1	21,5	21,1	22,4	26,6	21,3	21,0	27,2	27,2	27,2	27,2	21,0
80	20	100	INF	38,5	32,8	32,8	32,9	32,8	32,8	36,5	33,0	33,2	34,0	38,5	32,8	32,8	38,9	38,9	38,9	38,9	32,8
20	40	100	INF	0,8	0,1	0,1	0,0	0,0	0,0	0,2	0,2	0,0	0,2	0,6	0,1	0,0	0,1	0,0	0,0	0,0	0,0
40	40	100	INF	8,4	2,2	2,1	2,0	0,7	0,7	6,3	2,5	0,9	2,8	8,4	2,2	0,7	0,4	0,4	0,4	0,4	0,4
60	40	100	INF	18,4	9,8	9,8	9,4	8,5	8,5	17,1	10,0	9,0	11,1	19,9	9,8	8,5	15,3	15,3	15,3	15,3	8,5
80	40	100	INF	30,5	19,6	19,6	19,4	18,8	18,8	29,0	19,6	18,9	21,5	30,7	19,6	18,8	27,9	27,9	27,9	27,9	18,8
20	60	100	INF	0,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1	0,0	0,0	0,1	0,0	0,0	0,0	0,0
40	60	100	INF	5,1	0,5	0,5	0,5	0,0	0,0	3,5	0,5	0,0	0,8	5,0	0,5	0,0	0,2	0,1	0,0	0,0	0,0
60	60	100	INF	13,6	2,4	2,4	2,4	0,7	0,7	11,8	2,7	0,8	4,1	13,9	2,4	0,7	0,4	0,4	0,4	0,4	0,4
80	60	100	INF	22,8	9,4	9,4	9,1	7,6	7,6	22,7	9,7	7,8	11,9	23,5	9,4	7,6	14,7	14,7	14,7	14,7	7,6
20	80	100	INF	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
40	80	100	INF	2,4	0,1	0,1	0,1	0,0	0,0	2,2	0,1	0,0	0,2	2,4	0,1	0,0	0,1	0,0	0,0	0,0	0,0
60	80	100	INF	9,6	1,6	1,6	0,8	0,1	0,1	9,4	1,7	0,4	2,0	10,6	1,6	0,1	0,2	0,1	0,1	0,1	0,1
80	80	100	INF	18,2	3,1	3,1	2,6	0,8	0,8	18,1	3,4	1,4	5,6	17,9	3,1	0,8	0,4	0,4	0,4	0,4	0,4
20	20	600	INF	2,9	1,5	1,4	1,3	0,6	0,6	2,3	1,7	1,0	1,2	3,5	1,5	0,6	0,4	0,4	0,4	0,4	0,4
40	20	600	INF	13,5	10,0	10,0	9,9	9,5	9,5	11,8	10,4	9,6	10,5	14,7	10,0	9,5	13,4	13,4	13,4	13,4	9,5
60	20	600	INF	26,2	21,3	21,3	21,1	21,0	21,0	22,9	21,9	21,3	21,9	26,7	21,3	21,0	27,2	27,2	27,2	27,2	21,0
80	20	600	INF	37,8	32,8	32,8	32,8	32,8	32,8	34,0	33,0	32,8	33,6	37,9	32,8	32,8	38,9	38,9	38,9	38,9	32,8
20	40	600	INF	0,6	0,1	0,1	0,0	0,0	0,0	0,1	0,1	0,0	0,1	0,6	0,1	0,0	0,1	0,0	0,0	0,0	0,0
40	40	600	INF	8,0	2,2	2,0	1,7	0,7	0,7	3,6	2,4	1,0	2,5	7,8	2,2	0,7	0,4	0,4	0,4	0,4	0,4
60	40	600	INF	19,0	9,8	9,7	9,3	8,5	8,5	13,4	10,2	8,8	11,2	19,5	9,8	8,5	15,3	15,3	15,3	15,3	8,5
80	40	600	INF	29,6	19,6	19,5	19,4	18,8	18,8	24,4	19,6	19,0	21,6	30,4	19,6	18,8	27,9	27,9	27,9	27,9	18,8
20	60	600	INF	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,2	0,0	0,0	0,1	0,0	0,0	0,0	0,0
40	60	600	INF	4,2	0,5	0,4	0,4	0,0	0,0	1,5	0,6	0,0	0,6	4,7	0,5	0,0	0,2	0,1	0,0	0,0	0,0
60	60	600	INF	12,8	2,4	2,4	2,3	0,7	0,7	8,2	2,9	1,2	4,3	14,0	2,4	0,7	0,4	0,4	0,4	0,4	0,4
80	60	600	INF	23,5	9,4	9,4	8,8	7,6	7,6	17,5	9,5	8,2	12,6	23,2	9,4	7,6	14,7	14,7	14,7	14,7	7,6
20	80	600	INF	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
40	80	600	INF	2,3	0,1	0,1	0,1	0,0	0,0	0,8	0,1	0,0	0,2	2,6	0,1	0,0	0,1	0,0	0,0	0,0	0,0
60	80	600	INF	8,3	1,6	1,5	0,7	0,1	0,1	5,7	1,7	0,2	2,1	10,0	1,6	0,1	0,2	0,1	0,1	0,1	0,1
80	80	600	INF	17,9	3,1	3,0	2,2	0,8	0,8	14,4	3,4	1,2	5,7	18,8	3,1	0,8	0,4	0,4	0,4	0,4	0,4
20	20	1100	INF	2,8	1,5	1,1	1,3	0,6	0,6	1,9	1,3	0,8	1,2	3,5	1,5	0,6	0,4	0,4	0,4	0,4	0,4
40	20	1100	INF	13,7	10,0	9,8	10,0	9,5	9,5	11,2	10,4	9,5	10,3	14,8	10,0	9,5	13,4	13,4	13,4	13,4	9,5
60	20	1100	INF	25,9	21,3	21,3	21,2	21,0	21,0	22,6	21,7	21,4	21,8	26,0	21,3	21,0	27,2	27,2	27,2	27,2	21,0
80	20	1100	INF	37,8	32,8	32,7	32,7	32,8	32,8	34,0	33,2	33,0	33,9	38,2	32,8	32,8	38,9	38,9	38,9	38,9	32,8
20	40	1100	INF	0,6	0,1	0,1	0,0	0,0	0,0	0,1	0,1	0,0	0,1	1,2	0,1	0,0	0,1	0,0	0,0	0,0	0,0
40	40	1100	INF	7,8	2,2	1,9	1,6	0,7	0,7	4,1	2,5	1,3	2,5	7,9	2,2	0,7	0,4	0,4	0,4	0,4	0,4

Table A.6: Objective Function Values - Cont'd

T	W	Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
60	40	1100	INF	18,1	9,8	9,4	9,5	8,5	8,5	12,1	9,8	8,6	11,1	16,7	9,8	8,5	15,3	15,3	15,3	15,3	8,5
80	40	1100	INF	29,5	19,6	19,6	19,3	18,8	18,8	23,4	19,8	19,0	21,5	28,8	19,6	18,8	27,9	27,9	27,9	27,9	18,8
20	60	1100	INF	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1	0,0	0,0	0,1	0,0	0,0	0,0	0,0
40	60	1100	INF	4,3	0,5	0,4	0,4	0,0	0,0	0,9	0,5	0,1	0,8	4,7	0,5	0,0	0,2	0,1	0,0	0,0	0,0
60	60	1100	INF	12,7	2,4	2,1	2,3	0,7	0,6	6,7	2,6	0,8	4,3	13,5	2,4	0,7	0,4	0,4	0,4	0,4	0,4
80	60	1100	INF	21,8	9,4	9,3	8,7	7,6	7,6	16,7	9,7	8,1	11,8	23,2	9,4	7,6	14,7	14,7	14,7	14,7	7,6
20	80	1100	INF	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
40	80	1100	INF	2,2	0,1	0,1	0,0	0,0	0,0	0,5	0,1	0,0	0,2	2,6	0,1	0,0	0,1	0,0	0,0	0,0	0,0
60	80	1100	INF	8,9	1,6	1,4	0,7	0,1	0,1	3,7	1,6	0,1	1,8	9,3	1,6	0,1	0,2	0,1	0,1	0,1	0,1
80	80	1100	INF	18,1	3,1	3,0	2,2	0,8	0,8	12,5	3,1	1,1	5,5	17,9	3,1	0,8	0,4	0,4	0,4	0,4	0,4
20	20	1600	INF	2,8	1,5	0,9	1,1	0,6	0,6	1,8	1,5	0,8	1,3	3,9	1,4	0,6	0,4	0,4	0,4	0,4	0,4
40	20	1600	INF	13,5	10,0	9,8	9,8	9,5	9,5	10,8	10,0	9,8	10,5	13,4	10,0	9,5	13,4	13,4	13,4	13,4	9,5
60	20	1600	INF	25,9	21,3	21,3	21,1	21,0	21,0	22,2	21,3	21,4	21,8	26,2	21,3	21,0	27,2	27,2	27,2	27,2	21,0
80	20	1600	INF	37,9	32,8	32,8	32,7	32,8	32,8	33,9	33,3	33,0	33,2	37,2	32,8	32,8	38,9	38,9	38,9	38,9	32,8
20	40	1600	INF	0,5	0,1	0,1	0,0	0,0	0,0	0,0	0,1	0,0	0,1	1,0	0,1	0,0	0,1	0,0	0,0	0,0	0,0
40	40	1600	INF	7,8	2,2	1,6	1,6	0,7	0,7	4,2	2,2	1,0	2,8	7,6	2,2	0,7	0,4	0,4	0,4	0,4	0,4
60	40	1600	INF	18,2	9,8	9,2	9,4	8,5	8,5	12,4	10,0	8,6	11,3	18,0	9,8	8,5	15,3	15,3	15,3	15,3	8,5
80	40	1600	INF	29,6	19,6	19,5	19,3	18,8	18,8	22,4	19,9	19,0	21,1	30,0	19,6	18,8	27,9	27,9	27,9	27,9	18,8
20	60	1600	INF	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1	0,0	0,0	0,1	0,0	0,0	0,0	0,0
40	60	1600	INF	4,3	0,5	0,4	0,4	0,0	0,0	1,2	0,4	0,1	0,6	4,3	0,5	0,0	0,2	0,1	0,0	0,0	0,0
60	60	1600	INF	12,9	2,4	2,1	2,1	0,7	0,7	6,4	2,5	0,8	4,5	13,5	2,4	0,7	0,4	0,4	0,4	0,4	0,4
80	60	1600	INF	22,7	9,4	9,2	8,7	7,6	7,6	15,8	9,8	8,0	12,2	22,6	9,4	7,6	14,7	14,7	14,7	14,7	7,6
20	80	1600	INF	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
40	80	1600	INF	1,9	0,1	0,1	0,0	0,0	0,0	0,5	0,1	0,0	0,2	2,8	0,1	0,0	0,1	0,0	0,0	0,0	0,0
60	80	1600	INF	9,2	1,6	1,5	0,7	0,1	0,1	3,3	1,8	0,1	1,6	9,9	1,6	0,1	0,2	0,1	0,1	0,1	0,1
80	80	1600	INF	18,3	3,1	2,9	2,5	0,8	0,8	10,8	3,2	0,9	5,5	18,1	3,1	0,8	0,4	0,4	0,4	0,4	0,4
20	20	2100	INF	2,6	1,5	0,8	1,2	0,6	0,6	1,9	1,9	0,8	1,1	3,4	1,5	0,6	0,4	0,4	0,4	0,4	0,4
40	20	2100	INF	13,6	10,0	9,8	9,8	9,5	9,5	11,4	10,3	9,8	10,4	14,7	10,0	9,5	13,4	13,4	13,4	13,4	9,5
60	20	2100	INF	25,8	21,3	21,2	21,1	21,0	21,0	22,3	21,7	21,4	21,9	25,7	21,3	21,0	27,2	27,2	27,2	27,2	21,0
80	20	2100	INF	37,9	32,8	32,7	32,7	32,8	32,8	34,2	33,3	33,3	33,4	38,0	32,8	32,8	38,9	38,9	38,9	38,9	32,8
20	40	2100	INF	0,6	0,1	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,1	0,7	0,1	0,0	0,1	0,0	0,0	0,0	0,0
40	40	2100	INF	7,8	2,2	1,8	1,6	0,7	0,7	4,0	2,2	1,1	2,7	7,6	2,2	0,7	0,4	0,4	0,4	0,4	0,4
60	40	2100	INF	18,1	9,8	9,3	9,2	8,5	8,5	12,1	9,7	8,5	11,2	18,1	9,8	8,5	15,3	15,3	15,3	15,3	8,5
80	40	2100	INF	29,1	19,6	19,4	19,3	18,8	18,8	23,1	20,2	19,0	21,3	30,9	19,6	18,8	27,9	27,9	27,9	27,9	18,8
20	60	2100	INF	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1	0,0	0,0	0,1	0,0	0,0	0,0	0,0
40	60	2100	INF	3,8	0,5	0,4	0,3	0,0	0,0	0,7	0,5	0,0	0,6	5,0	0,5	0,0	0,2	0,1	0,0	0,0	0,0
60	60	2100	INF	12,5	2,4	2,0	2,2	0,7	0,6	5,7	2,6	0,9	4,1	12,4	2,4	0,7	0,4	0,4	0,4	0,4	0,4
80	60	2100	INF	22,8	9,4	9,2	8,8	7,6	7,6	13,4	9,5	8,0	12,0	23,3	9,4	7,6	14,7	14,7	14,7	14,7	7,6

Table A.7: Objective Function Values - Cont'd

T	W	Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	80	2100	INF	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
40	80	2100	INF	1,9	0,1	0,1	0,0	0,0	0,0	0,4	0,1	0,0	0,2	2,6	0,1	0,0	0,1	0,0	0,0	0,0	0,0
60	80	2100	INF	8,7	1,6	1,4	0,7	0,1	0,1	2,7	1,6	0,2	1,8	9,2	1,6	0,1	0,2	0,1	0,1	0,1	0,1
80	80	2100	INF	18,1	3,1	2,9	2,2	0,8	0,8	9,9	3,2	1,1	5,8	19,2	3,1	0,8	0,4	0,4	0,4	0,4	0,4
20	20	2600	INF	2,9	1,5	0,9	1,2	0,6	0,6	1,5	1,3	0,8	1,1	3,5	1,5	0,6	0,4	0,4	0,4	0,4	0,4
40	20	2600	INF	13,6	10,0	9,6	9,8	9,5	9,5	11,1	10,1	9,9	10,5	14,1	10,0	9,5	13,4	13,4	13,4	13,4	9,5
60	20	2600	INF	24,9	21,3	21,2	21,1	21,0	21,0	22,4	21,6	21,2	21,8	26,4	21,3	21,0	27,2	27,2	27,2	27,2	21,0
80	20	2600	INF	37,4	32,8	32,8	32,6	32,8	32,8	33,9	33,2	33,1	33,8	37,9	32,8	32,8	38,9	38,9	38,9	38,9	32,8
20	40	2600	INF	0,4	0,1	0,0	0,0	0,0	0,0	0,1	0,1	0,0	0,1	1,0	0,1	0,0	0,1	0,0	0,0	0,0	0,0
40	40	2600	INF	7,8	2,2	1,5	1,4	0,7	0,7	3,7	2,2	1,1	2,7	8,5	2,2	0,7	0,4	0,4	0,4	0,4	0,4
60	40	2600	INF	18,1	9,8	9,5	9,4	8,5	8,5	11,7	10,2	8,7	10,9	19,0	9,8	8,5	15,3	15,3	15,3	15,3	8,5
80	40	2600	INF	29,4	19,6	19,4	19,3	18,8	18,8	23,0	19,6	19,2	21,1	29,4	19,6	18,8	27,9	27,9	27,9	27,9	18,8
20	60	2600	INF	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,2	0,0	0,0	0,1	0,0	0,0	0,0	0,0
40	60	2600	INF	4,3	0,5	0,4	0,4	0,0	0,0	0,8	0,5	0,1	0,7	5,0	0,5	0,0	0,2	0,1	0,0	0,0	0,0
60	60	2600	INF	12,8	2,4	1,8	2,3	0,7	0,7	5,6	2,8	0,7	4,3	13,1	2,4	0,7	0,4	0,4	0,4	0,4	0,4
80	60	2600	INF	22,7	9,4	9,2	8,8	7,6	7,6	13,4	9,8	8,0	12,0	23,8	9,4	7,6	14,7	14,7	14,7	14,7	7,6
20	80	2600	INF	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
40	80	2600	INF	1,8	0,1	0,1	0,0	0,0	0,0	0,3	0,1	0,0	0,2	2,8	0,1	0,0	0,1	0,0	0,0	0,0	0,0
60	80	2600	INF	8,7	1,6	1,2	0,7	0,1	0,1	2,4	1,8	0,1	1,8	9,3	1,6	0,1	0,2	0,1	0,1	0,1	0,1
6	6	100	INF	0,4	0,7	0,7	0,6	0,4	0,4	0,7	0,6	0,6	0,3	0,6	0,7	0,4	0,3	0,3	0,3	0,3	0,3
9	6	100	INF	1,9	1,8	1,8	1,7	1,7	1,6	1,8	2,3	1,8	1,6	2,0	1,8	1,7	2,4	2,4	2,4	1,6	1,7
6	9	600	INF	0,2	0,3	0,3	0,2	0,1	0,1	0,2	0,3	0,3	0,1	0,2	0,1	0,1	0,2	0,2	0,2	0,2	0,1
9	6	600	INF	1,7	1,8	1,7	1,7	1,7	1,6	2,0	1,7	1,6	1,6	2,5	1,8	1,7	2,4	2,4	2,4	1,6	1,7
9	9	600	INF	1,1	1,0	1,0	0,9	0,6	0,6	1,2	1,3	0,6	0,6	1,1	1,0	0,6	0,5	0,5	0,5	0,5	0,5
6	9	1100	INF	0,1	0,3	0,2	0,1	0,1	0,1	0,2	0,2	0,1	0,1	0,6	0,2	0,1	0,2	0,2	0,2	0,2	0,1
9	9	1100	INF	0,9	1,0	0,8	0,7	0,6	0,6	1,1	0,8	0,7	0,6	1,0	1,0	0,6	0,5	0,5	0,5	0,5	0,5
9	6	1100	INF	1,7	1,8	1,7	1,7	1,7	1,6	1,8	1,7	1,7	1,6	2,1	1,8	1,7	2,4	2,4	2,4	1,6	1,7
6	9	1600	INF	0,1	0,3	0,2	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,4	0,3	0,1	0,2	0,2	0,2	0,2	0,1
9	9	1600	INF	0,8	1,0	0,7	0,7	0,6	0,6	0,8	0,8	0,7	0,5	1,1	1,0	0,6	0,5	0,5	0,5	0,5	0,5
6	9	2100	INF	0,1	0,3	0,2	0,1	0,1	0,1	0,1	0,2	0,2	0,1	0,3	0,3	0,1	0,2	0,2	0,2	0,2	0,1
9	9	2100	INF	0,7	1,0	0,8	0,8	0,6	0,6	0,9	0,8	0,7	0,5	1,0	0,9	0,6	0,5	0,5	0,5	0,5	0,5
6	9	2600	INF	0,1	0,3	0,2	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,3	0,2	0,1	0,2	0,2	0,2	0,2	0,1
9	9	2600	INF	0,9	1,0	0,6	0,8	0,6	0,5	0,7	0,8	0,7	0,6	1,0	1,0	0,6	0,5	0,5	0,5	0,5	0,5
6	9	2600	INF	0,1	0,3	0,2	0,1	0,1	0,1	0,1	0,2	0,1	0,1	0,3	0,2	0,1	0,2	0,2	0,2	0,2	0,1
9	9	2600	INF	0,8	1,0	0,7	0,7	0,6	0,6	0,8	0,9	0,6	0,6	1,2	1,0	0,6	0,5	0,5	0,5	0,5	0,5
80	80	2600	INF	17,8	3,1	2,8	2,4	0,8	0,8	9,3	3,1	1,2	5,6	18,0	3,1	0,8	0,4	0,4	0,4	0,4	0,4
3	3	100	0,3	0,3	0,5	0,5	0,3	0,4	0,3	0,5	0,4	0,3	0,3	0,5	0,4	0,3	0,3	0,3	0,3	0,3	0,3
6	3	100	1,9	1,9	1,9	1,9	1,9	1,9	1,9	2,1	2,0	1,9	1,9	2,0	1,9	1,9	1,9	1,9	1,9	1,9	1,9

Table A.8: Objective Function Values - Cont'd

T	W	Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
9	3	100	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	4,1	3,7	3,7	4,1	4,1	4,1	3,7	3,7
3	6	100	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0
3	9	100	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
3	3	600	0,3	0,3	0,5	0,3	0,3	0,4	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3
6	3	600	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9
9	3	600	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	4,1	3,7	3,7	4,1	4,1	4,1	3,7	3,7
3	6	600	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
6	6	600	0,3	0,3	0,7	0,7	0,4	0,4	0,3	0,6	0,7	0,3	0,3	0,6	0,5	0,4	0,3	0,3	0,3	0,3	0,3
3	9	600	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
3	3	1100	0,3	0,3	0,5	0,3	0,3	0,4	0,3	0,3	0,3	0,3	0,3	0,5	0,3	0,4	0,3	0,3	0,3	0,3	0,3
6	3	1100	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	2,1	1,9	1,9	1,9	1,9	1,9	1,9	1,9
9	3	1100	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	4,3	3,7	3,7	4,1	4,1	4,1	3,7	3,7
3	6	1100	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
6	6	1100	0,3	0,3	0,7	0,5	0,4	0,4	0,3	0,3	0,5	0,6	0,3	0,4	0,5	0,4	0,3	0,3	0,3	0,3	0,3
3	9	1100	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
3	3	1600	0,3	0,3	0,5	0,3	0,3	0,4	0,3	0,3	0,3	0,3	0,3	0,3	0,4	0,3	0,3	0,3	0,3	0,3	0,3
6	3	1600	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9
9	3	1600	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	4,0	3,7	3,7	4,1	4,1	4,1	3,7	3,7
3	6	1600	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0
6	6	1600	0,3	0,3	0,7	0,5	0,5	0,4	0,3	0,4	0,5	0,4	0,3	0,9	0,6	0,4	0,3	0,3	0,3	0,3	0,3
9	6	1600	1,6	1,7	1,8	1,7	1,7	1,7	1,6	1,6	1,7	1,6	1,6	2,0	1,8	1,7	2,4	2,4	2,4	1,6	1,6
3	9	1600	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
3	3	2100	0,3	0,3	0,5	0,3	0,3	0,4	0,3	0,3	0,3	0,3	0,3	0,3	0,5	0,3	0,3	0,3	0,3	0,3	0,3
6	3	2100	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9
9	3	2100	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	4,0	3,7	3,7	4,1	4,1	4,1	3,7	3,7
3	6	2100	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
6	6	2100	0,3	0,3	0,7	0,5	0,4	0,4	0,3	0,4	0,3	0,3	0,3	1,0	0,5	0,4	0,3	0,3	0,3	0,3	0,3
9	6	2100	1,6	1,7	1,8	1,7	1,7	1,7	1,6	1,7	1,6	1,7	1,6	1,9	1,8	1,6	2,4	2,4	2,4	1,6	1,6
3	9	2100	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
3	3	2600	0,3	0,3	0,5	0,3	0,3	0,4	0,3	0,3	0,3	0,3	0,3	0,5	0,4	0,3	0,3	0,3	0,3	0,3	0,3
6	3	2600	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	1,9	2,6	1,9	1,9	1,9	1,9	1,9	1,9	1,9
9	3	2600	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	3,7	4,1	4,1	4,1	3,7	3,7
3	6	2600	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
6	6	2600	0,3	0,3	0,7	0,5	0,4	0,4	0,3	0,4	0,3	0,5	0,3	1,0	0,4	0,4	0,3	0,3	0,3	0,3	0,3
9	6	2600	1,6	1,7	1,8	1,7	1,7	1,7	1,6	1,8	1,7	1,7	1,6	2,2	1,8	1,7	2,4	2,4	2,4	1,6	1,6
3	9	2600	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	AVG:		INF	8,6	5,0	4,8	4,7	4,4	4,4	6,6	5,0	4,5	5,4	8,9	4,9	4,4	6,0	5,9	5,9	5,9	**4,37**

Table A.9: Deviations

T	W	Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	20	100	INF	3,06	1,10	1,10	0,92	0,22	0,22	2,03	1,47	0,27	1,07	3,04	1,10	0,22	0,00	0,00	0,00	0,00	0,00
40	20	100	INF	4,78	0,55	0,45	0,65	0,00	0,00	3,35	0,73	0,63	1,49	5,12	0,55	0,00	3,94	3,94	3,94	3,94	0,00
60	20	100	INF	5,63	0,31	0,31	0,31	0,00	0,00	3,14	0,58	0,18	1,40	5,69	0,31	0,00	6,21	6,21	6,21	6,21	0,00
80	20	100	INF	5,76	0,07	0,07	0,14	0,00	0,00	3,78	0,27	0,42	1,22	5,78	0,07	0,00	6,12	6,12	6,12	6,12	0,00
20	40	100	INF	0,86	0,17	0,16	0,07	0,00	0,00	0,23	0,20	0,00	0,20	0,67	0,17	0,00	0,14	0,02	0,01	0,01	0,00
40	40	100	INF	7,91	1,71	1,65	1,57	0,24	0,24	5,87	2,08	0,42	2,39	7,99	1,71	0,24	0,00	0,00	0,00	0,00	0,00
60	40	100	INF	9,88	1,28	1,28	0,91	0,00	0,00	8,60	1,47	0,44	2,54	11,40	1,28	0,00	6,81	6,81	6,81	6,81	0,00
80	40	100	INF	11,69	0,74	0,74	0,56	0,00	0,00	10,17	0,78	0,11	2,71	11,90	0,74	0,00	9,10	9,10	9,10	9,10	0,00
20	60	100	INF	0,23	0,01	0,01	0,01	0,00	0,00	0,07	0,01	0,00	0,02	0,15	0,01	0,00	0,14	0,00	0,00	0,00	0,00
40	60	100	INF	5,04	0,45	0,45	0,45	0,00	0,00	3,50	0,43	0,01	0,74	4,94	0,45	0,00	0,14	0,04	0,02	0,02	0,00
60	60	100	INF	13,22	1,97	1,95	1,97	0,23	0,23	11,37	2,26	0,42	3,72	13,48	1,97	0,23	0,00	0,00	0,00	0,00	0,00
80	60	100	INF	15,21	1,76	1,76	1,46	0,00	0,00	15,06	2,10	0,16	4,31	15,87	1,76	0,00	7,08	7,08	7,08	7,08	0,00
20	80	100	INF	0,04	0,00	0,00	0,00	0,00	0,00	0,02	0,00	0,00	0,00	0,07	0,00	0,00	0,04	0,00	0,00	0,00	0,00
40	80	100	INF	2,47	0,14	0,14	0,10	0,00	0,00	2,20	0,15	0,03	0,27	2,45	0,14	0,00	0,15	0,01	0,00	0,00	0,00
60	80	100	INF	9,59	1,56	1,56	0,75	0,00	0,00	9,32	1,67	0,33	1,94	10,51	1,56	0,00	0,16	0,09	0,08	0,08	0,00
80	80	100	INF	17,84	2,67	2,67	2,16	0,41	0,41	17,68	3,03	1,00	5,25	17,49	2,67	0,41	0,00	0,00	0,00	0,00	0,00
20	20	600	INF	2,48	1,10	1,00	0,91	0,22	0,19	1,89	1,26	0,60	0,81	3,12	1,08	0,22	0,00	0,00	0,00	0,00	0,00
40	20	600	INF	4,03	0,55	0,50	0,45	0,00	0,00	2,35	0,91	0,14	1,01	5,26	0,55	0,00	3,94	3,94	3,94	3,94	0,00
60	20	600	INF	5,29	0,31	0,31	0,11	0,00	0,00	1,96	0,90	0,34	0,90	5,77	0,31	0,00	6,21	6,21	6,21	6,21	0,00
80	20	600	INF	5,08	0,07	0,07	0,02	0,00	0,00	1,24	0,27	0,06	0,89	5,16	0,07	0,00	6,12	6,12	6,12	6,12	0,00
20	40	600	INF	0,60	0,17	0,13	0,09	0,00	0,00	0,17	0,14	0,02	0,14	0,67	0,17	0,00	0,14	0,02	0,01	0,01	0,00
40	40	600	INF	7,55	1,71	1,55	1,22	0,24	0,24	3,16	1,99	0,52	2,09	7,36	1,71	0,24	0,00	0,00	0,00	0,00	0,00
60	40	600	INF	10,48	1,31	1,22	0,84	0,03	0,00	4,96	1,74	0,35	2,71	11,04	1,31	0,03	6,84	6,84	6,84	6,84	0,03
80	40	600	INF	10,78	0,74	0,73	0,56	0,00	0,00	5,63	0,78	0,24	2,75	11,56	0,74	0,00	9,10	9,10	9,10	9,10	0,00
20	60	600	INF	0,18	0,01	0,01	0,01	0,00	0,00	0,01	0,01	0,00	0,02	0,26	0,01	0,00	0,14	0,00	0,00	0,00	0,00
80	40	600	INF	10,78	0,74	0,73	0,56	0,00	0,00	5,63	0,78	0,24	2,75	11,56	0,74	0,00	9,10	9,10	9,10	9,10	0,00
20	60	600	INF	0,18	0,01	0,01	0,01	0,00	0,00	0,01	0,01	0,00	0,02	0,26	0,01	0,00	0,14	0,00	0,00	0,00	0,00
40	60	600	INF	4,20	0,45	0,42	0,38	0,00	0,00	1,47	0,55	0,01	0,55	4,67	0,45	0,00	0,14	0,04	0,02	0,02	0,00
60	60	600	INF	12,39	1,97	1,97	1,84	0,23	0,23	7,75	2,44	0,74	3,88	13,61	1,97	0,23	0,00	0,00	0,00	0,00	0,00
80	60	600	INF	15,87	1,76	1,76	1,19	0,00	0,00	9,83	1,85	0,58	4,94	15,55	1,76	0,00	7,08	7,08	7,08	7,08	0,00
20	80	600	INF	0,05	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,04	0,00	0,00	0,04	0,00	0,00	0,00	0,00
40	80	600	INF	2,38	0,14	0,13	0,10	0,00	0,00	0,85	0,14	0,01	0,25	2,69	0,14	0,00	0,15	0,01	0,00	0,00	0,00
60	80	600	INF	8,27	1,56	1,48	0,70	0,00	0,00	5,66	1,63	0,19	2,07	9,91	1,56	0,00	0,16	0,09	0,08	0,08	0,00
80	80	600	INF	17,48	2,67	2,64	1,85	0,41	0,41	14,05	2,96	0,77	5,29	18,43	2,67	0,41	0,00	0,00	0,00	0,00	0,00
20	20	1100	INF	2,39	1,10	0,71	0,88	0,22	0,22	1,44	0,89	0,34	0,77	3,06	1,10	0,22	0,00	0,00	0,00	0,00	0,00
40	20	1100	INF	4,28	0,55	0,31	0,50	0,00	0,00	1,71	0,97	0,00	0,86	5,38	0,55	0,00	3,94	3,94	3,94	3,94	0,00
60	20	1100	INF	4,98	0,31	0,31	0,23	0,00	0,00	1,60	0,70	0,46	0,88	5,09	0,31	0,00	6,21	6,21	6,21	6,21	0,00
80	20	1100	INF	5,05	0,11	0,00	0,02	0,03	0,03	1,24	0,52	0,32	1,20	5,50	0,11	0,03	6,15	6,15	6,15	6,15	0,03

Table A.10: Deviations - Cont'd

T	W	Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	40	1100	INF	0,64	0,17	0,13	0,07	0,00	0,00	0,12	0,09	0,00	0,14	1,21	0,17	0,00	0,14	0,02	0,01	0,01	0,00
40	40	1100	INF	7,35	1,71	1,48	1,17	0,24	0,22	3,63	2,05	0,89	2,11	7,45	1,71	0,24	0,00	0,00	0,00	0,00	0,00
60	40	1100	INF	9,58	1,28	0,88	1,00	0,00	0,00	3,57	1,25	0,14	2,60	8,16	1,28	0,00	6,81	6,81	6,81	6,81	0,00
80	40	1100	INF	10,66	0,74	0,74	0,49	0,00	0,00	4,54	1,04	0,19	2,69	10,04	0,74	0,00	9,10	9,10	9,10	9,10	0,00
20	60	1100	INF	0,15	0,01	0,01	0,01	0,00	0,00	0,01	0,01	0,00	0,02	0,17	0,01	0,00	0,14	0,00	0,00	0,00	0,00
40	60	1100	INF	4,24	0,45	0,38	0,37	0,00	0,00	0,90	0,46	0,09	0,73	4,65	0,45	0,00	0,14	0,04	0,02	0,02	0,00
60	60	1100	INF	12,23	1,97	1,63	1,84	0,23	0,21	6,27	2,21	0,38	3,84	13,09	1,97	0,23	0,00	0,00	0,00	0,00	0,00
80	60	1100	INF	14,13	1,76	1,72	1,12	0,00	0,00	9,08	2,03	0,43	4,21	15,61	1,76	0,00	7,08	7,08	7,08	7,08	0,00
20	80	1100	INF	0,04	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,05	0,00	0,00	0,04	0,00	0,00	0,00	0,00
40	80	1100	INF	2,23	0,14	0,13	0,08	0,00	0,00	0,55	0,13	0,00	0,21	2,64	0,14	0,00	0,15	0,01	0,00	0,00	0,00
60	80	1100	INF	8,86	1,56	1,32	0,65	0,00	0,00	3,64	1,59	0,00	1,75	9,23	1,56	0,00	0,16	0,09	0,08	0,08	0,00
80	80	1100	INF	17,72	2,67	2,58	1,85	0,41	0,41	12,12	2,68	0,76	5,15	17,51	2,67	0,41	0,00	0,00	0,00	0,00	0,00
20	20	1600	INF	2,42	1,10	0,48	0,72	0,22	0,19	1,43	1,06	0,40	0,87	3,44	0,94	0,22	0,00	0,00	0,00	0,00	0,00
40	20	1600	INF	4,00	0,55	0,38	0,36	0,00	0,00	1,30	0,50	0,31	1,02	3,94	0,55	0,00	3,94	3,94	3,94	3,94	0,00
60	20	1600	INF	4,92	0,31	0,31	0,12	0,00	0,00	1,25	0,31	0,40	0,88	5,23	0,31	0,00	6,21	6,21	6,21	6,21	0,00
80	20	1600	INF	5,21	0,16	0,16	0,00	0,09	0,09	1,24	0,63	0,33	0,56	4,58	0,16	0,09	6,20	6,20	6,20	6,20	0,09
20	40	1600	INF	0,59	0,17	0,11	0,08	0,00	0,00	0,08	0,10	0,00	0,10	0,99	0,16	0,00	0,14	0,02	0,01	0,01	0,00
40	40	1600	INF	7,33	1,71	1,19	1,19	0,24	0,23	3,77	1,76	0,60	2,32	7,15	1,71	0,24	0,00	0,00	0,00	0,00	0,00
60	40	1600	INF	9,68	1,28	0,70	0,86	0,00	0,00	3,84	1,52	0,14	2,82	9,51	1,28	0,00	6,81	6,81	6,81	6,81	0,00
80	40	1600	INF	10,78	0,74	0,66	0,46	0,00	0,00	3,59	1,06	0,24	2,31	11,19	0,74	0,00	9,10	9,10	9,10	9,10	0,00
20	60	1600	INF	0,14	0,01	0,01	0,01	0,00	0,00	0,00	0,01	0,00	0,02	0,16	0,01	0,00	0,14	0,00	0,00	0,00	0,00
40	60	1600	INF	4,26	0,45	0,36	0,35	0,00	0,00	1,15	0,40	0,08	0,63	4,33	0,45	0,00	0,14	0,04	0,02	0,02	0,00
60	60	1600	INF	12,45	1,97	1,70	1,67	0,23	0,23	5,94	2,11	0,34	4,03	13,06	1,97	0,23	0,00	0,00	0,00	0,00	0,00
80	60	1600	INF	15,13	1,78	1,63	1,06	0,01	0,00	8,23	2,18	0,38	4,61	15,03	1,78	0,01	7,09	7,09	7,09	7,09	0,01
20	80	1600	INF	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,04	0,00	0,00	0,04	0,00	0,00	0,00	0,00
40	80	1600	INF	1,97	0,14	0,12	0,07	0,00	0,00	0,57	0,15	0,00	0,23	2,79	0,14	0,00	0,15	0,01	0,00	0,00	0,00
60	80	1600	INF	9,11	1,56	1,40	0,62	0,00	0,00	3,27	1,72	0,07	1,58	9,86	1,56	0,00	0,16	0,09	0,08	0,08	0,00
80	80	1600	INF	17,90	2,67	2,50	2,09	0,41	0,41	10,37	2,86	0,54	5,15	17,69	2,67	0,41	0,00	0,00	0,00	0,00	0,00
20	20	2100	INF	2,19	1,10	0,37	0,75	0,22	0,19	1,50	1,46	0,37	0,73	2,98	1,10	0,22	0,00	0,00	0,00	0,00	0,00
40	20	2100	INF	4,17	0,55	0,34	0,36	0,00	0,00	1,98	0,83	0,29	0,96	5,21	0,55	0,00	3,94	3,94	3,94	3,94	0,00
60	20	2100	INF	4,88	0,31	0,26	0,12	0,00	0,00	1,34	0,77	0,49	0,93	4,73	0,31	0,00	6,21	6,21	6,21	6,21	0,00
80	20	2100	INF	5,17	0,11	0,00	0,03	0,03	0,03	1,48	0,54	0,59	0,66	5,28	0,11	0,03	6,15	6,15	6,15	6,15	0,03
20	40	2100	INF	0,68	0,17	0,09	0,08	0,00	0,00	0,07	0,05	0,00	0,09	0,78	0,17	0,00	0,14	0,02	0,01	0,01	0,00
40	40	2100	INF	7,40	1,71	1,33	1,18	0,24	0,22	3,54	1,71	0,69	2,29	7,13	1,71	0,24	0,00	0,00	0,00	0,00	0,00
60	40	2100	INF	9,63	1,28	0,83	0,65	0,00	0,00	3,60	1,23	0,02	2,65	9,62	1,28	0,00	6,81	6,81	6,81	6,81	0,00
80	40	2100	INF	10,28	0,74	0,57	0,46	0,00	0,00	4,32	1,38	0,18	2,44	12,13	0,74	0,00	9,10	9,10	9,10	9,10	0,00
20	60	2100	INF	0,13	0,01	0,01	0,01	0,00	0,00	0,01	0,00	0,00	0,01	0,19	0,01	0,00	0,14	0,00	0,00	0,00	0,00
40	60	2100	INF	3,82	0,45	0,40	0,33	0,00	0,00	0,70	0,44	0,00	0,61	5,00	0,45	0,00	0,14	0,04	0,02	0,02	0,00

Table A.11: Deviations - Cont'd

T	W	Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
60	60	2100	INF	12,05	1,97	1,56	1,77	0,23	0,21	5,25	2,17	0,46	3,71	11,97	1,97	0,23	0,00	0,00	0,00	0,00	0,00
80	60	2100	INF	15,16	1,78	1,60	1,18	0,01	0,00	5,78	1,91	0,36	4,42	15,65	1,78	0,01	7,09	7,09	7,09	7,09	0,01
20	80	2100	INF	0,02	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,04	0,00	0,00	0,04	0,00	0,00	0,00	0,00
40	80	2100	INF	1,98	0,14	0,13	0,07	0,00	0,00	0,40	0,16	0,01	0,25	2,68	0,14	0,00	0,15	0,01	0,00	0,00	0,00
60	80	2100	INF	8,65	1,56	1,37	0,64	0,00	0,00	2,68	1,57	0,11	1,74	9,12	1,56	0,00	0,16	0,09	0,08	0,08	0,00
80	80	2100	INF	17,75	2,67	2,55	1,80	0,41	0,36	9,47	2,84	0,72	5,38	18,83	2,67	0,41	0,00	0,00	0,00	0,00	0,00
20	20	2600	INF	2,52	1,10	0,51	0,75	0,22	0,17	1,12	0,91	0,37	0,67	3,05	1,10	0,22	0,00	0,00	0,00	0,00	0,00
40	20	2600	INF	4,16	0,55	0,18	0,36	0,00	0,00	1,63	0,66	0,44	1,01	4,68	0,55	0,00	3,94	3,94	3,94	3,94	0,00
60	20	2600	INF	3,96	0,31	0,26	0,14	0,00	0,00	1,42	0,68	0,27	0,86	5,47	0,31	0,00	6,21	6,21	6,21	6,21	0,00
80	20	2600	INF	4,72	0,19	0,19	0,00	0,11	0,11	1,22	0,54	0,48	1,20	5,23	0,19	0,11	6,23	6,23	6,23	6,23	0,11
20	40	2600	INF	0,40	0,17	0,06	0,08	0,00	0,00	0,10	0,09	0,00	0,11	1,05	0,17	0,00	0,14	0,02	0,01	0,01	0,00
40	40	2600	INF	7,37	1,71	1,05	0,92	0,24	0,22	3,28	1,75	0,68	2,21	8,10	1,71	0,24	0,00	0,00	0,00	0,00	0,00
60	40	2600	INF	9,57	1,31	1,01	0,93	0,03	0,00	3,19	1,70	0,19	2,42	10,47	1,31	0,03	6,84	6,84	6,84	6,84	0,03
80	40	2600	INF	10,60	0,79	0,67	0,54	0,05	0,00	4,24	0,86	0,47	2,38	10,64	0,79	0,05	9,15	9,15	9,15	9,15	0,05
20	60	2600	INF	0,13	0,01	0,01	0,01	0,00	0,00	0,01	0,01	0,00	0,01	0,24	0,01	0,00	0,14	0,00	0,00	0,00	0,00
40	60	2600	INF	4,32	0,45	0,39	0,36	0,00	0,00	0,81	0,53	0,04	0,71	4,98	0,45	0,00	0,14	0,04	0,02	0,02	0,00
60	60	2600	INF	12,40	1,97	1,41	1,84	0,23	0,23	5,22	2,38	0,31	3,86	12,72	1,97	0,23	0,00	0,00	0,00	0,00	0,00
80	60	2600	INF	15,14	1,78	1,59	1,20	0,01	0,00	5,74	2,21	0,38	4,35	16,19	1,78	0,01	7,09	7,09	7,09	7,09	0,01
20	80	2600	INF	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,03	0,00	0,00	0,04	0,00	0,00	0,00	0,00
40	80	2600	INF	1,85	0,14	0,11	0,08	0,00	0,00	0,36	0,14	0,00	0,23	2,88	0,14	0,00	0,15	0,01	0,00	0,00	0,00
60	80	2600	INF	8,62	1,56	1,12	0,63	0,00	0,00	2,32	1,78	0,02	1,74	9,21	1,56	0,00	0,16	0,09	0,08	0,08	0,00
6	6	100	INF	0,07	0,45	0,45	0,29	0,07	0,07	0,37	0,30	0,33	0,00	0,31	0,45	0,07	0,00	0,00	0,00	0,00	0,00
9	6	100	INF	0,26	0,14	0,14	0,06	0,06	0,00	0,12	0,64	0,20	0,00	0,35	0,14	0,06	0,77	0,77	0,77	0,00	0,06
6	9	100	INF	0,09	0,16	0,16	0,06	0,00	0,00	0,07	0,16	0,24	0,00	0,12	0,01	0,00	0,13	0,09	0,07	0,07	0,00
9	9	100	INF	0,61	0,50	0,50	0,42	0,15	0,15	0,70	0,81	0,15	0,09	0,58	0,50	0,15	0,00	0,00	0,00	0,00	0,00
9	6	600	INF	0,01	0,14	0,07	0,03	0,06	0,00	0,32	0,09	0,00	0,00	0,81	0,14	0,06	0,77	0,77	0,77	0,00	0,06
6	9	600	INF	0,01	0,16	0,11	0,04	0,00	0,00	0,05	0,07	0,00	0,00	0,47	0,05	0,00	0,13	0,09	0,07	0,07	0,00
9	9	600	INF	0,39	0,50	0,28	0,21	0,15	0,15	0,59	0,36	0,18	0,09	0,50	0,50	0,15	0,00	0,00	0,00	0,00	0,00
9	6	1100	INF	0,01	0,14	0,01	0,01	0,06	0,00	0,15	0,08	0,06	0,00	0,47	0,14	0,06	0,77	0,77	0,77	0,00	0,06
6	9	1100	INF	0,00	0,16	0,10	0,04	0,00	0,00	0,02	0,00	0,01	0,00	0,33	0,16	0,00	0,13	0,09	0,07	0,07	0,00
9	9	1100	INF	0,31	0,50	0,27	0,21	0,15	0,10	0,32	0,30	0,26	0,03	0,62	0,50	0,10	0,00	0,00	0,00	0,00	0,00
6	9	1600	INF	0,04	0,16	0,06	0,04	0,00	0,00	0,03	0,06	0,06	0,00	0,18	0,16	0,00	0,13	0,09	0,07	0,07	0,00
9	9	1600	INF	0,22	0,50	0,30	0,32	0,15	0,10	0,45	0,31	0,21	0,03	0,57	0,40	0,15	0,00	0,00	0,00	0,00	0,00
6	9	2100	INF	0,02	0,16	0,06	0,03	0,00	0,00	0,01	0,04	0,00	0,00	0,15	0,14	0,00	0,13	0,09	0,07	0,07	0,00
9	9	2100	INF	0,41	0,50	0,08	0,29	0,15	0,03	0,23	0,31	0,27	0,08	0,51	0,50	0,15	0,00	0,00	0,00	0,00	0,00
6	9	2600	INF	0,00	0,16	0,10	0,04	0,00	0,00	0,00	0,06	0,00	0,00	0,20	0,11	0,00	0,13	0,09	0,07	0,07	0,00
9	9	2600	INF	0,32	0,50	0,21	0,21	0,15	0,10	0,35	0,47	0,15	0,09	0,75	0,50	0,15	0,00	0,00	0,00	0,00	0,00
80	80	2600	INF	17,45	2,67	2,42	1,97	0,41	0,40	8,90	2,70	0,76	5,18	17,64	2,67	0,41	0,00	0,00	0,00	0,00	0,00

Table A.12: Deviations - Cont'd

T	W	Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
3	3	100	0,00	0,00	0,19	0,19	0,00	0,09	0,00	0,19	0,09	0,00	0,00	0,19	0,09	0,00	0,00	0,00	0,00	0,00	0,00
6	3	100	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,26	0,18	0,00	0,00	0,18	0,00	0,00	0,00	0,00	0,00	0,00	0,00
9	3	100	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,01	0,01	0,01	0,00	0,43	0,00	0,00	0,46	0,46	0,46	0,00	0,00
3	6	100	0,00	0,00	0,03	0,03	0,00	0,00	0,00	0,00	0,03	0,00	0,00	0,13	0,03	0,00	0,02	0,00	0,00	0,00	0,00
3	9	100	0,00	0,00	0,04	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,04	0,01	0,00	0,00	0,01	0,00	0,00	0,00
3	3	600	0,00	0,00	0,19	0,00	0,00	0,09	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
6	3	600	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
9	3	600	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,42	0,00	0,00	0,46	0,46	0,46	0,00	0,00
3	6	600	0,00	0,00	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,01	0,00	0,00	0,02	0,00	0,00	0,00	0,00
6	6	600	0,00	0,00	0,45	0,37	0,13	0,07	0,00	0,28	0,37	0,00	0,00	0,28	0,22	0,07	0,00	0,00	0,00	0,00	0,00
3	9	600	0,00	0,00	0,04	0,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,01	0,01	0,00	0,00	0,01	0,00	0,00	0,00
3	3	1100	0,00	0,00	0,19	0,00	0,00	0,09	0,00	0,00	0,00	0,00	0,00	0,25	0,00	0,09	0,00	0,00	0,00	0,00	0,00
6	3	1100	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,26	0,00	0,00	0,00	0,00	0,00	0,00	0,00
9	3	1100	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,65	0,00	0,00	0,46	0,46	0,46	0,00	0,00
3	6	1100	0,00	0,00	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,01	0,00	0,02	0,00	0,00	0,00	0,00
6	6	1100	0,00	0,00	0,45	0,24	0,13	0,07	0,00	0,00	0,16	0,31	0,00	0,07	0,19	0,07	0,00	0,00	0,00	0,00	0,00
3	9	1100	0,00	0,00	0,04	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,01	0,00	0,00	0,00	0,01	0,00	0,00	0,00
3	3	1600	0,00	0,00	0,19	0,00	0,00	0,09	0,00	0,00	0,00	0,00	0,00	0,00	0,09	0,00	0,00	0,00	0,00	0,00	0,00
6	3	1600	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
9	3	1600	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,32	0,00	0,00	0,46	0,46	0,46	0,00	0,00
3	6	1600	0,00	0,00	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,10	0,03	0,00	0,02	0,00	0,00	0,00	0,00
6	6	1600	0,00	0,00	0,45	0,24	0,19	0,07	0,00	0,13	0,24	0,07	0,00	0,65	0,28	0,07	0,00	0,00	0,00	0,00	0,00
9	6	1600	0,00	0,01	0,14	0,01	0,01	0,06	0,00	0,00	0,07	0,00	0,00	0,33	0,14	0,06	0,77	0,77	0,77	0,00	0,00
3	9	1600	0,00	0,00	0,04	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,01	0,00	0,00	0,00	0,01	0,00	0,00	0,00
3	3	2100	0,00	0,00	0,19	0,00	0,00	0,09	0,00	0,00	0,00	0,00	0,00	0,00	0,19	0,00	0,00	0,00	0,00	0,00	0,00
6	3	2100	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
9	3	2100	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,34	0,00	0,00	0,46	0,46	0,46	0,00	0,00
3	6	2100	0,00	0,00	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,05	0,03	0,00	0,02	0,00	0,00	0,00	0,00
6	6	2100	0,00	0,00	0,45	0,24	0,13	0,07	0,00	0,13	0,00	0,00	0,00	0,65	0,22	0,07	0,00	0,00	0,00	0,00	0,00
9	6	2100	0,00	0,01	0,14	0,07	0,01	0,06	0,00	0,08	0,00	0,03	0,00	0,26	0,14	0,00	0,77	0,77	0,77	0,00	0,00
3	9	2100	0,00	0,00	0,04	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,01	0,01	0,00	0,00	0,01	0,00	0,00	0,00
3	3	2600	0,00	0,00	0,19	0,00	0,00	0,09	0,00	0,00	0,00	0,00	0,00	0,19	0,09	0,00	0,00	0,00	0,00	0,00	0,00
6	3	2600	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,74	0,00	0,00	0,00	0,00	0,00	0,00	0,00
9	3	2600	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,46	0,46	0,46	0,00	0,00
3	6	2600	0,00	0,00	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,01	0,03	0,00	0,02	0,00	0,00	0,00	0,00
9	6	2600	0,00	0,03	0,14	0,07	0,03	0,06	0,00	0,13	0,03	0,01	0,00	0,57	0,14	0,06	0,77	0,77	0,77	0,00	0,00
3	9	2600	0,00	0,00	0,04	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,02	0,01	0,00	0,00	0,01	0,00	0,00	0,00
AVG:			INF	4,27	0,64	0,53	0,43	0,06	0,05	2,27	0,71	0,19	1,11	4,59	0,62	0,06	1,66	1,63	1,63	1,58	**0,004**

Table A.13: Ranks

T	W	Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	20	100	12	11	7	6	4	2	2	9	8	3	5	10	7	2	1	1	1	1	1
40	20	100	12	10	3	2	5	1	1	8	6	4	7	11	3	1	9	9	9	9	1
60	20	100	10	7	3	3	3	1	1	6	4	2	5	8	3	1	9	9	9	9	1
80	20	100	11	8	2	2	3	1	1	7	4	5	6	9	2	1	10	10	10	10	1
20	40	100	14	13	8	7	5	1	1	11	10	2	9	12	8	1	6	4	3	3	1
40	40	100	12	10	6	5	4	2	2	9	7	3	8	11	6	2	1	1	1	1	1
60	40	100	11	9	4	4	3	1	1	8	5	2	6	10	4	1	7	7	7	7	1
80	40	100	11	9	4	4	3	1	1	8	5	2	6	10	4	1	7	7	7	7	1
20	60	100	14	13	8	7	6	1	1	10	5	3	9	12	8	1	11	4	2	2	1
40	60	100	13	12	8	7	8	1	1	10	6	2	9	11	8	1	5	4	3	3	1
60	60	100	11	9	5	4	5	2	2	8	6	3	7	10	5	2	1	1	1	1	1
80	60	100	11	9	4	4	3	1	1	8	5	2	6	10	4	1	7	7	7	7	1
20	80	100	14	12	8	7	5	1	1	10	6	3	9	13	8	1	11	4	2	2	1
40	80	100	13	12	6	6	5	1	1	10	7	4	9	11	6	1	8	3	2	2	1
60	80	100	13	11	7	7	6	1	1	10	8	5	9	12	7	1	4	3	2	2	1
80	80	100	11	10	5	5	4	2	2	9	6	3	7	8	5	2	1	1	1	1	1
20	20	600	14	12	9	7	6	3	2	11	10	4	5	13	8	3	1	1	1	1	1
40	20	600	12	10	5	4	3	1	1	8	6	2	7	11	5	1	9	9	9	9	1
60	20	600	11	8	3	3	2	1	1	7	5	4	6	9	3	1	10	10	10	10	1
80	20	600	11	8	4	4	2	1	1	7	5	3	6	9	4	1	10	10	10	10	1
20	40	600	14	12	10	6	5	1	1	11	9	4	7	13	10	1	8	3	2	2	1
40	40	600	12	11	6	5	4	2	2	9	7	3	8	10	6	2	1	1	1	1	1
60	40	600	13	11	6	5	4	2	1	9	7	3	8	12	6	2	10	10	10	10	2
80	40	600	12	10	5	4	3	1	1	8	6	2	7	11	5	1	9	9	9	9	1
20	60	600	15	13	10	9	6	2	1	7	8	4	11	14	10	2	12	5	3	3	2
40	60	600	14	12	8	7	6	1	1	11	9	2	10	13	8	1	5	4	3	3	1
60	60	600	11	9	5	5	4	2	2	8	6	3	7	10	5	2	1	1	1	1	1
80	60	600	11	10	4	4	3	1	1	8	5	2	6	9	4	1	7	7	7	7	1
20	80	600	14	13	8	7	5	1	1	10	6	2	9	11	8	1	12	4	3	3	1
40	80	600	14	12	7	6	5	1	1	11	8	3	10	13	7	1	9	4	2	2	1
60	80	600	14	12	8	7	6	1	1	11	9	5	10	13	8	1	4	3	2	2	1
80	80	600	12	10	6	5	4	2	2	9	7	3	8	11	6	2	1	1	1	1	1
20	20	1100	12	10	8	4	6	2	2	9	7	3	5	11	8	2	1	1	1	1	1
40	20	1100	11	9	4	2	3	1	1	7	6	1	5	10	4	1	8	8	8	8	1
60	20	1100	11	8	3	3	2	1	1	7	5	4	6	9	3	1	10	10	10	10	1
80	20	1100	12	9	4	1	2	3	3	8	6	5	7	10	4	3	11	11	11	11	3
20	40	1100	14	12	11	8	5	1	1	7	6	2	9	13	11	1	10	4	3	3	1
40	40	1100	13	11	7	6	5	3	2	10	8	4	9	12	7	3	1	1	1	1	1

Table A.14: Ranks - Cont'd

T	**W**	**Time**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
60	40	1100	12	11	6	3	4	1	1	8	5	2	7	10	6	1	9	9	9	9	1
80	40	1100	12	11	5	4	3	1	1	8	6	2	7	10	5	1	9	9	9	9	1
20	60	1100	14	12	9	7	5	1	1	8	6	2	10	13	9	1	11	4	3	3	1
40	60	1100	14	12	8	7	6	1	1	11	9	4	10	13	8	1	5	3	2	2	1
60	60	1100	13	11	7	5	6	3	2	10	8	4	9	12	7	3	1	1	1	1	1
80	60	1100	12	10	5	4	3	1	1	9	6	2	7	11	5	1	8	8	8	8	1
20	80	1100	15	13	10	7	6	1	1	11	5	3	8	14	9	1	12	4	2	2	1
40	80	1100	14	12	8	6	5	1	1	11	7	2	10	13	8	1	9	4	3	3	1
60	80	1100	14	12	8	7	6	1	1	11	9	2	10	13	8	1	5	4	3	3	1
80	80	1100	12	11	6	5	4	2	2	9	7	3	8	10	6	2	1	1	1	1	1
20	20	1600	14	12	10	5	6	3	2	11	9	4	7	13	8	3	1	1	1	1	1
40	20	1600	12	11	6	4	3	1	1	8	5	2	7	10	6	1	9	9	9	9	1
60	20	1600	10	7	3	3	2	1	1	6	3	4	5	8	3	1	9	9	9	9	1
80	20	1600	11	9	3	3	1	2	2	7	6	4	5	8	3	2	10	10	10	10	2
20	40	1600	15	13	12	9	6	1	1	5	7	2	8	14	11	1	10	4	3	3	1
40	40	1600	13	12	7	6	5	3	2	10	8	4	9	11	7	3	1	1	1	1	1
60	40	1600	12	11	5	3	4	1	1	8	6	2	7	10	5	1	9	9	9	9	1
80	40	1600	12	10	5	4	3	1	1	8	6	2	7	11	5	1	9	9	9	9	1
20	60	1600	14	11	9	6	7	1	1	5	8	2	10	13	9	1	12	4	3	3	1
40	60	1600	14	12	9	7	6	1	1	11	8	4	10	13	9	1	5	3	2	2	1
60	60	1600	12	10	6	5	4	2	2	9	7	3	8	11	6	2	1	1	1	1	1
80	60	1600	13	12	6	5	4	2	1	10	7	3	8	11	6	2	9	9	9	9	2
20	80	1600	14	11	10	8	5	1	1	7	6	3	9	13	10	1	12	4	2	2	1
40	80	1600	14	12	7	6	5	1	1	11	8	2	10	13	7	1	9	4	3	3	1
60	80	1600	14	12	8	7	6	1	1	11	10	2	9	13	8	1	5	4	3	3	1
80	80	1600	12	11	6	5	4	2	2	9	7	3	8	10	6	2	1	1	1	1	1
20	20	2100	13	11	8	5	7	3	2	10	9	4	6	12	8	3	1	1	1	1	1
40	20	2100	12	10	5	3	4	1	1	8	6	2	7	11	5	1	9	9	9	9	1
60	20	2100	12	10	4	3	2	1	1	8	6	5	7	9	4	1	11	11	11	11	1
80	20	2100	12	9	4	1	2	3	3	8	5	6	7	10	4	3	11	11	11	11	3
20	40	2100	15	13	12	8	7	1	1	6	5	2	9	14	11	1	10	4	3	3	1
40	40	2100	12	11	7	6	5	3	2	9	7	4	8	10	7	3	1	1	1	1	1
60	40	2100	12	11	6	4	3	1	1	8	5	2	7	10	6	1	9	9	9	9	1
80	40	2100	12	10	5	4	3	1	1	8	6	2	7	11	5	1	9	9	9	9	1
20	60	2100	14	11	10	6	7	1	1	8	5	3	9	13	10	1	12	4	2	2	1
40	60	2100	14	12	9	7	6	1	1	11	8	2	10	13	9	1	5	4	3	3	1
60	60	2100	13	12	7	5	6	3	2	10	8	4	9	11	7	3	1	1	1	1	1
80	60	2100	13	11	6	5	4	2	1	9	7	3	8	12	6	2	10	10	10	10	2

Table A.15: Ranks - Cont'd

T	**W**	**Time**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
20	80	2100	15	12	11	8	6	1	1	7	5	3	9	14	10	1	13	4	2	2	1
40	80	2100	14	12	7	6	5	1	1	11	9	3	10	13	7	1	8	4	2	2	1
60	80	2100	14	12	8	7	6	1	1	11	9	4	10	13	8	1	5	3	2	2	1
80	80	2100	13	11	7	6	5	3	2	10	8	4	9	12	7	3	1	1	1	1	1
20	20	2600	13	11	9	5	7	3	2	10	8	4	6	12	9	3	1	1	1	1	1
40	20	2600	12	10	5	2	3	1	1	8	6	4	7	11	5	1	9	9	9	9	1
60	20	2600	12	9	5	3	2	1	1	8	6	4	7	10	5	1	11	11	11	11	1
80	20	2600	11	8	3	3	1	2	2	7	5	4	6	9	3	2	10	10	10	10	2
20	40	2600	15	13	12	6	7	2	1	9	8	3	10	14	12	2	11	5	4	4	2
40	40	2600	13	11	7	6	5	3	2	10	8	4	9	12	7	3	1	1	1	1	1
60	40	2600	13	11	6	5	4	2	1	9	7	3	8	12	6	2	10	10	10	10	2
80	40	2600	13	11	6	5	4	2	1	9	7	3	8	12	6	2	10	10	10	10	2
20	60	2600	14	11	9	8	6	1	1	5	7	2	10	13	9	1	12	4	3	3	1
40	60	2600	14	12	8	7	6	1	1	11	9	3	10	13	8	1	5	4	2	2	1
60	60	2600	12	10	6	4	5	2	2	9	7	3	8	11	6	2	1	1	1	1	1
80	60	2600	13	11	6	5	4	2	1	9	7	3	8	12	6	2	10	10	10	10	2
20	80	2600	15	12	11	9	8	2	1	7	6	4	10	13	11	2	14	5	3	3	2
40	80	2600	15	13	9	7	6	2	1	12	8	3	11	14	9	2	10	5	4	4	2
60	80	2600	14	12	8	7	6	1	1	11	10	2	9	13	8	1	5	4	3	3	1
80	80	2600	13	11	7	6	5	3	2	10	8	4	9	12	7	3	1	1	1	1	1
6	6	100	9	2	8	8	3	2	2	7	4	6	1	5	8	2	1	1	1	1	1
6	9	100	15	7	13	11	4	2	2	5	12	14	1	9	3	2	10	8	6	6	2
9	9	100	10	7	5	5	4	3	3	8	9	3	2	6	5	3	1	1	1	1	1
9	6	600	12	3	8	6	4	5	1	9	7	1	1	11	8	5	10	10	10	2	5
6	9	600	14	3	12	10	4	2	2	6	8	2	1	13	5	2	11	9	7	7	2
9	9	600	12	8	9	6	5	3	3	11	7	4	2	10	9	3	1	1	1	1	1
6	9	1100	13	2	11	9	6	2	1	5	3	4	1	12	11	2	10	8	7	7	2
9	9	1100	13	9	11	7	5	4	3	10	8	6	2	12	11	3	1	1	1	1	1
9	6	1100	10	3	6	3	3	4	1	7	5	4	1	8	6	4	9	9	9	2	4
6	9	1600	14	4	12	8	5	2	1	3	6	7	1	13	12	2	11	10	9	9	2
6	9	2100	14	4	13	7	5	2	2	3	6	2	1	12	11	2	10	9	8	8	2
9	9	2100	13	10	11	3	8	5	2	6	9	7	4	12	11	5	1	1	1	1	1
9	9	1600	14	6	12	7	9	4	3	11	8	5	2	13	10	4	1	1	1	1	1
6	9	2600	13	1	11	8	4	2	1	2	5	3	1	12	9	2	10	7	6	6	2
9	9	2600	11	6	9	5	5	4	3	7	8	4	2	10	9	4	1	1	1	1	1
9	6	100	11	7	5	5	3	3	1	4	9	6	1	8	5	3	10	10	10	2	3
3	3	100	1	1	3	3	1	2	1	3	2	1	1	3	2	1	1	1	1	1	1
6	3	100	1	1	1	1	1	1	1	4	3	1	1	3	1	1	1	1	1	2	1

Table A.16: Ranks - Cont'd

T	**W**	**Time**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
9	3	100	1	1	1	1	1	1	1	3	4	3	1	5	1	1	6	6	6	2	1
3	6	100	1	1	5	5	1	2	2	1	4	1	1	6	5	2	3	1	1	1	1
3	9	100	1	1	11	9	5	2	1	2	6	2	3	10	8	2	4	7	4	4	1
3	3	600	1	1	3	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1
6	3	600	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1
9	3	600	1	1	1	1	1	1	1	1	1	1	1	3	1	1	4	4	4	2	1
3	6	600	1	1	6	3	1	2	2	1	1	1	1	4	1	2	5	1	1	1	1
6	6	600	1	1	9	7	3	2	1	5	8	1	1	6	4	2	1	1	1	1	1
3	9	600	1	1	10	6	3	2	1	2	5	1	1	9	8	1	4	7	4	4	1
3	3	1100	1	1	3	1	1	2	1	1	1	1	1	4	1	2	1	1	1	1	1
6	3	1100	1	1	1	1	1	1	1	1	1	1	1	3	1	1	1	1	1	2	1
9	3	1100	1	1	1	1	1	1	1	1	1	1	1	4	1	1	3	3	3	2	1
3	6	1100	1	1	6	3	1	2	2	1	1	1	1	1	4	2	5	1	1	1	1
6	6	1100	1	1	8	6	3	2	1	1	4	7	1	2	5	2	1	1	1	1	1
3	9	1100	1	1	9	6	3	2	1	1	2	1	1	8	5	2	4	7	4	4	1
3	3	1600	1	1	3	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1
6	3	1600	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1
9	3	1600	1	1	1	1	1	1	1	1	1	1	1	3	1	1	4	4	4	2	1
3	6	1600	1	1	5	3	1	2	2	1	1	1	1	6	5	2	4	1	1	1	1
6	6	1600	1	1	8	6	4	2	1	3	5	2	1	9	7	2	1	1	1	1	1
9	6	1600	1	3	6	3	3	4	1	1	5	1	1	7	6	4	8	8	8	2	1
3	9	1600	1	1	9	3	1	2	1	1	5	1	1	8	6	2	4	7	4	4	1
3	3	2100	1	1	3	1	1	2	1	1	1	1	1	1	3	1	1	1	1	1	1
6	3	2100	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1
9	3	2100	1	1	1	1	1	1	1	1	1	1	1	3	1	1	4	4	4	2	1
3	6	2100	1	1	5	3	1	2	2	1	1	1	1	6	5	2	4	1	1	1	1
6	6	2100	1	1	6	5	3	2	1	3	1	1	1	7	4	2	1	1	1	1	1
9	6	2100	1	3	8	6	3	5	1	7	1	4	1	9	8	1	10	10	10	2	1
3	9	2100	1	1	9	4	3	2	1	1	1	1	1	7	8	2	5	6	5	5	1
3	3	2600	1	1	3	1	1	2	1	1	1	1	1	3	2	1	1	1	1	1	1
6	3	2600	1	1	1	1	1	1	1	1	1	1	1	3	1	1	1	1	1	2	1
9	3	2600	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3	3	3	2	1
3	6	2600	1	1	6	3	1	2	2	1	1	1	1	4	6	2	5	1	1	1	1
6	6	2600	1	1	6	5	3	2	1	2	1	4	1	7	2	2	1	1	1	1	1
9	6	2600	1	4	8	6	4	5	1	7	4	3	1	9	8	5	10	10	10	2	1
3	9	2600	1	1	8	3	1	2	1	1	1	1	1	7	6	2	4	5	4	4	1
	AVG:		10	8	7	5	4	2	1	7	5,7	3	5,5	10	6	2	5,9	5	4	4	**1,2**