STONY BROOK UNIVERSITY, NEW YORK

RESEARCH PROFICIENCY EXAMINATION REPORT

Department of Computer Science

On the Complexity of Allocation Problems with Probabilistic Players

Rishab Nithyanand

Committee:
Professor Rob Johnson (Advisor)
Professor Michael Bender
Professor Steven Skiena

Summer 2012

TABLE OF CONTENTS

ABSTRACT

On the Complexity of Allocation Problems with Probabilistic Players

By

Rishab Nithyanand

Department of Computer Science

Resource Allocation Problems (*RAP*) are central to operations research and business models. In this report we focus on a class of RAP called probabilistic allocation problems. Traditional allocation problems deal with assigning a given set of resources to a number of tasks in order to optimize an objective cost function. Probabilistic allocation problems, on the other hand, deal with assigning resources to tasks where allocations may have probabilistic costs associated with them. The goal is to optimize the expected value of the objective cost function. Such problems, appear more complicated than traditional allocation problems due to the presence of an inseparable objective cost function that is typically non-linear.

In this report we provide a detailed analysis of the complexity of several probabilistic allocation problems. In particular, we focus on the Weapons-Target Allocation (WTA) problem in its single round and multiple round guises, and the Password Allocation (*PA*) problem. We discuss the current state-of-the-art with respect to exact and approximate algorithms for solving these problems.

The PA problem is a new combinatorial optimization problem that may be defined as follows: Given n accounts – $a_1, ..., a_n$, where each account (a_i) has an associated value (v_i) and compromise probability ($q_i = 1 - p_i$) . Is there a way to allocate k passwords (potentially, of varying strengths) to these n accounts (where, $k \leq n$) such that the Expected Gain (EG) is maximized (or, conversely – the Expected Loss (EL) is minimized)? This problem models many real world parallel job/resource allocation problems, and the results presented here are applicable in many domains including parallel process scheduling, job-machine allocations, etc. We study the complexity of the *PA* problem and several of its variants. In particular, the *PA* problem and its variants are shown to be NP Hard, and special polynomial time solvable cases are identified. Finally, several heuristic approaches to solving the problem are identified.

Chapter 1

Theoretical Foundations

In simplest terms, allocation problems deal with allocating resources to tasks (all of which are to be completed) while optimizing a given cost function. Such problems can be mathematically described by numbering the resources ($i = 1, ..., n$) and tasks ($j = 1, ..., k$, where $n \geq k$) and introducing a binary assignment matrix $X = \{x_{ij}\}$ having the following meaning:

$$x_{ij} = \begin{cases} 1 & \text{if resource } i \text{ is assigned to task } j \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

Then, if c_{ij} is the cost of using resource i to complete task j, our problem is to select from among all the assignment matrices which satisfy the constraints given by:

$$\sum_{i=1}^{n} x_{ij} \geq 1, \ \forall j \in \{1, ..., k\} \tag{1.2}$$

$$\sum_{j=1}^{k} x_{ij} \leq 1, \ \forall i \in \{1, ..., n\} \tag{1.3}$$

the one matrix which maximizes/minimizes the given objective function, for eg.,

$$\sum_{i=0}^{n} \sum_{j=0}^{k} x_{ij} c_{ij}$$

The constraints given above ensure that each task is allocated at least one resource and that each resource is allocated to at most one task. If the number of resources and tasks to be matched are equal – i.e., $n = k$, the inequalities in the above constraints no longer exist, and the problem is called an *assignment problem*.

Besides the representation described in equations (1.1) – (1.3), it is also possible to represent allocation problems in the form of bipartite graphs. By definition, a bipartite graph G is one with disjoint vertex sets U and V, and an edge set E such that all edges in E have one end point in U and another in V. Consider a subset of E which has every vertex of G occurring in it at least once. Any such subset which contains all the vertices of one of the disjoint vertex sets exactly once (say U), and the vertices in the other disjoint set (say V) at least once, is a valid allocation between a set of tasks (represented by V) and resources (represented by U). It should be clear that the maximum size of such an edge set is equal to the number resources to be allocated, and the minimum size of such

an edge set is the number of tasks requiring resources. When using graphs to represent an allocation problem, our goal is to find an edge set (such as the one just described), which has optimal cost.

Naive approaches to solve this problem that involve generating every possible allocation matrix (or, edge set) are prohibitively expensive – i.e., they require time that is exponential in the number of resources to solve. Therefore specialized algorithms are required to solve any useful (non-trivial) instances of such allocation problems. The existence of such algorithms, however, depends on the complexity of the problem to be solved. For example, some allocation problems such as the Linear Sum Assignment Problem (*LSAP*) are solvable in polynomial time – i.e., it belongs in the complexity class P. However, other problems such as the Generalized Assignment Problem (*GAP*) are known to be NP Hard, and as a result are expected to not have polynomial time algorithms to find exact solutions. In these cases, approximate algorithms are used to get near optimal solutions for large instances in reasonable time. Therefore, it is imperative to understand the complexity of such problems before designing algorithms to solve them. To this end, in this report we study the complexity of several allocation problems with probabilistic/stochastic parameters.

1.1 Non-Linear Programming and Convex Functions

Non-linear programming (*NLP*) is the process of solving a system of constraints, over a set of real variables, along with an objective function to be optimized. The constraints and the objective function may be non-linear [3]. Mathematically, an NLP problem may be defined as follows [21]:

$$minimize\ f(x) \tag{1.4}$$

$$subject\ to:\ \ g_i(x) \geq 0, \forall i \in \{i = 1, ..., m\} \tag{1.5}$$

$$and:\ \ h_j(x) = 0, \forall j \in \{j = 1, ..., k\} \tag{1.6}$$

where, the functions f, $g_1, ..., g_m$, $h_1, ..., h_k$: $\mathsf{R}^n \rightarrow \mathsf{R}$. f is called the *objective function*, while $g_1, ..., g_m$, $h_1, ..., h_k$ are the constraints. The set of inputs $X = \{x\}$ that satisfy equations 1.5 and 1.6 belong to the *feasible region* of the NLP problem and any single point $x \in X$ is called a *feasible solution*.

We call a set C in the euclidean space R^n a *convex set* if for any pair of points (x, y) and $\alpha \in [0, 1]$, $(1-\alpha)x + \alpha y \in C$. In simpler terms, a set is a convex set if for every pair of points (x, y) within the euclidean space containing the set, every point on the line segment connecting x and y is also contained in the set. A function f is called a *convex function* if the set given by the epigraph of the function (i.e., the set of points on or above the graph of the function) is a convex set. Simply put, any continuous function where the graph of the function lies below the line segment connecting any two points on the graph is a convex function. An example is illustrated in Figure 1.1. Further, we call a convex function that has $f(x) > -\infty$ for every x and $f(x) < +\infty$ for some x, a *proper convex function*.

Let $B(x_0, \delta) = \{x \in \mathsf{R}^n | ||x - x_0|| \leq \delta\}$ for a given x_0 and δ where $||u|| = \sqrt{\sum_{j=1}^{n} u_j^2}$. We call any point $x' \in X$ a *local minimum* if there exists a $\delta > 0$ such that $f(x) \geq f(x')$ for all $x \in X \cap B(x', \delta)$. If $f(x) \geq f(x')$ for all $x \in X$, we call x' the *global minimum* or an *optimal solution*. A non-linear programming problem with the functions $f, -g_i$ being convex and h_j being affine possesses a convex set as its feasible region. Such problems are also known as *convex programming problems*.

Convex functions are particularly important in non-linear programming and optimization since they possess two very important properties – (1) any local minimum of a convex function is also a global minimum and therefore,

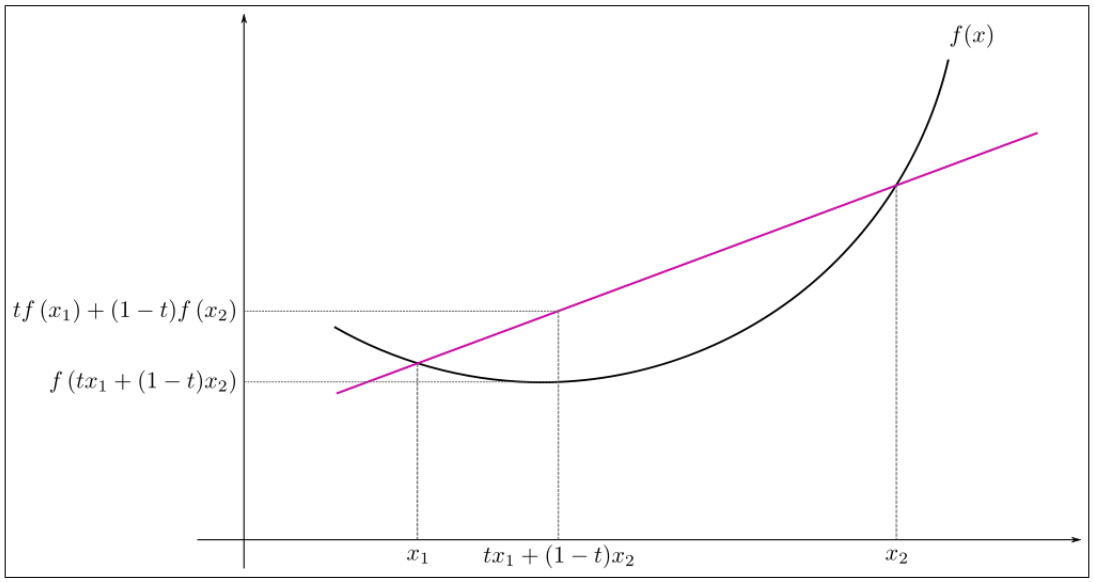

Figure 1.1: The graph of a convex function (source: Wikimedia Commons)

if $f'(c) = 0$, then c is a global minimum (i.e., optimal solution) of the convex function, (2) the following inequality (*Jensen's inequality*) applies to the expectation of all convex functions: $E(f(X)) \geq f(E(X))$.

Theorem 1. *A local minimum of a convex function $f : \mathsf{R}^n \to \mathsf{R}$ is always a global minimum.*

Proof. By definition, suppose we have x as a local minimum for f, there is a continuous subset (U) of the feasible region (X) where $f(x) \leq f(y), \forall y \in U$. We need to show that $f(x) \leq f(y), \forall y \in X$. We define a *convex combination* of elements from a set $V = \{v_1, ..., v_n\}$ as a linear combination of the form:

$$\begin{aligned} &\lambda_1 v_1 + \lambda_2 v_2 + ... + \lambda_n v_n \\ &\textit{such that } : \sum_{i=0}^{n} \lambda_i = 1 \\ &\textit{and } \lambda_i \geq 0, \forall i \in \{1, ..., n\} \end{aligned}$$

Consider the convex combination given by $c = (1-t)x + ty, t \in (0,1)$. Since all multiplication and addition operations in R, R^n are continuous, as $t \to 0$, we have $c \to x$. Therefore, for small enough t, we have $c \in U$. Then,

$$f(x) \leq f(ty + (1-t)x) \tag{1.7}$$

$$f(x) \leq tf(y) + (1-t)f(x) \tag{1.8}$$

Equation 1.7 holds for any small $t > 0$, while equation 1.8 follows from 1.7 and the definition of convex functions. Re-arranging the terms in 1.8, it is easy to see that $f(x) \leq f(y)$. □

We say that *Slater's constraints* hold when there exists a point $x_0 \in \mathsf{R}^n$ such that $g_i(x_0) > 0$ and $h_j(x_0) = 0$. A seminal result due to Kuhn and Tucker (known as the Kuhn-Tucker Theorem) [26] proved that if a given convex programming problem has all differentiable constraint functions and objective function and satisfies Slater's constraints, then a feasible solution $x \in X$ is optimal iff there exists an m-dimensional vector μ and an k-dimensional

vector λ such that:

$$\nabla f(x) - \sum_{i=1}^{m} \mu_i \nabla g_i(x) - \sum_{j=1}^{k} \lambda_j \nabla h_j(x) = 0 \tag{1.9}$$

$$\mu_i g_i(x) = 0, \forall i \in \{1,...,m\} \tag{1.10}$$

$$\mu \geq 0 \tag{1.11}$$

These conditions are the *Kuhn-Tucker conditions*. The variables $\mu_1,...,\mu_m$, and $\lambda_1,...,\lambda_k$ are known as *the Lagrange Multipliers*. If a convex programming problem has a separable objective function and Slater's constraints are satisfied, it is efficiently solvable (i.e., it is not NPHard). Popular algorithms that find optimal solutions for such problems using the Kuhn-Tucker constraints are RANK [31] and the Bitran-Hax line search algorithm [4].

So far we have seen that allocations of divisible resources (where $x \in \mathsf{R}^n$) in order to optimize a convex separable function is possible in polynomial time. However, it is often the case that resources may not be divisible (where $x \in \mathsf{Z}^+$), for e.g., allocating humans, machines, etc. Further, the objective function may not be separable. Such allocation problems with integer variables are more complicated and are often not efficiently solvable. We now show that a very general version of the problem, called the Generalized Resource Allocation (GRA) problem, is NPHard (proof adapted from [20]). Mathematically, the GRA problem can be formulated as follows:

$$minimize: \ z(x) \ = \ f(x_1,...,x_n) \tag{1.12}$$

$$subject\ to: \ x_j \in \mathsf{Z}^+, \forall j \in \{1,...,n\} \tag{1.13}$$

$$and: \ \sum_{j=1}^{n} x_j = N \tag{1.14}$$

Theorem 2. *Generalized Resource Allocation (GRA) problem $\in$ NP Hard.*

Proof. The decision version of GRA may be defined as follows: Given integers k and N, does there exist an $X = (x_1,...,x_n)$ in the feasible region of $z(X)$ such that $z(X) \leq k$ and $\sum_{j=1}^{n} x_j = N$? Clearly $GRA \in \mathsf{NP}$ since it is easy to verify if a given X has $z(X) \leq k$. To show that $GRA \in \mathsf{NPHard}$, we show a reduction via the following variation of the *set partition* problem (known to be NPHard).

Set Partition (SP): Given an $m \times n$ 0-1 matrix A such that $\sum_{i=1}^{m} a_{ij} \geq 1, \forall j \in \{1,...,n\}$ and $\sum_{j=1}^{n} a_{ij} \geq 1, \forall i \in \{1,...,m\}$, an integer N, and a vector $X = (x_1, x_2, ..., x_n)$, does there exist a $m \times n$ 0-1 matrix $Y = \{y_{ij}\}$ such that $\sum_{j=1}^{n} a_{ij} y_{ij} x_j = N, \forall i \in \{1,...,m\}$ and $\sum_{i=1}^{m} a_{ij} y_{ij} = 1, \forall j \in \{1,...,n\}$ where $y_{ij} \in \{0,1\}$.

Given an instance of SP = (A, N, X), we construct an instance of GRA given by $(z(Y), N, k = 0)$ where

$$z(Y) \ = \sum_{i=1}^{m} \left(\left(\sum_{j=1}^{n} a_{ij} y_{ij} \right) - 1 \right)^2 \tag{1.15}$$

$$subject\ to: \ \sum_{j=1}^{n} a_{ij} y_{ij} x_j = N, \forall i \in \{1,...,m\} \tag{1.16}$$

Iff $z(Y) \leq 0$, then there is a valid set partition for the given instance of SP. The reduction from GRA to SP is done in $O(nm + \log N)$. Clearly the above transformation is polynomial in size of the inputs. Therefore, any efficient algorithm for the *GRA* problem can be used to solve the *SP* problem efficiently. Since $SP \in \mathsf{NPHard}$, we have that $GRA \in \mathsf{NPHard}$.

□

It is important to keep in mind that while the GRA problem is NPHard, there exist many special cases for

integer allocation problems with inseparable functions that are efficiently solvable.

1.2 Assignment Problems, Matchings, and the Marriage Theorem

Assignment problems are a type of allocation problem where an equal number of resources and tasks are to be allocated to each other. As with allocation problems, they may be represented as bipartite graphs where the set of all vertices $= U \cup V$ and $|U| = |V|$. Consider a bipartite graph $G = (U \cup V, E)$ where $|U| = |V| = n$. We say that a *matching* exists in G if there exists a subset of the edge set E such that every vertex of G meets at most one edge of the matching. If there exists a matching set M that contains every vertex of G then, M is called a *perfect matching*. Clearly, every valid *assignment* (i.e., allocation between an equal number of tasks and resources) may be represented by a perfect matching. An important question that follows is, given a bipartite graph G given by $(U \cup V, E)$, does there exist a perfect matching? This problem known as the *bipartite perfect matching problem* has played an important role in the theory of assignment problems.

A seminal theorem, titled *Halls Theorem* (better known as *The Marriage Theorem*), by Philip Hall in 1935 [15] proved that perfect matchings (i.e., valid assignments) existed as long as a single condition (known as *Hall's Condition*) was satisfied by the given bipartite graph. In simple words, *Hall's Condition* stated that for any subset of vertices of size k belonging to U, the number of neighboring vertices belonging to V should be $\geq k$.

Theorem 3. *[The Marriage Theorem] Let $G = (U \cup V, E)$ be a bipartite graph and $N(U)$ denote the set of neighbors of vertices $\in U$. There exists a perfect matching in G iff for all subsets U' of U: $|U'| \leq |N(U')|$.*

Proof.

- It is easy to see that Hall's condition is necessary – i.e., if there is a subset which has fewer neighbors than elements, at least one of the members of that subset cannot be matched with a neighbor, implying that the graph cannot have a perfect matching.

- Now, we use induction to show that this condition is sufficient for a perfect matching. If $|U| = |V| = 1$, the only vertex in U will have to be mapped to the only vertex in $|V|$ (unless $E = \emptyset$, which is not permitted). Assume that the theorem holds for all bipartite graphs G with $|U| = |V| = k$ and let $G' = (U' \cup V', E')$ be a bipartite graph with $|U'| = |V'| = k+1$. We assume that G' also satisfies Hall's condition. Now, consider the following two cases:

  1. $|U''| < |N(U'')|$ holds for all non-empty proper subsets U'' of U'. Now, we match some vertex i with a randomly chosen neighbor j. By deleting i, j and all the edges incident on them, we have a new graph $G^* = (U^* \cup V^*, E^*)$ with $|U^*| = |V^*| = k$. By induction, Hall's condition still applies in G^*. Therefore, G^* must contain a matching of size k (and G' must have a matching of $k+1$).
  2. $|U''| = |N(U'')|$ holds for some non-empty proper subset U'' of U'. Due to our induction, we can match each $i \in U''$ with an appropriate $j \in N(U'')$. Now, we delete the sets $U'' \in U'$ and $N(U'') \in V'$ and the incident edges. We get a new graph G^* and need to show that Hall's condition is satisfied by this graph. If it is not, then there is some subset W of U^* for which $|W| > |N_{G^*}(W)|$ – i.e., there is a subset W in G^* which has fewer neighbors than members. However, this implies that $|W \cup U''| > |N_{G^*}(W) \cup N(U'')|$, which is a contradiction.

Therefore, we have shown that Hall's condition is necessary and sufficient for the existence of a perfect matching. □

A follow up to the marriage theorem is the following observation: Every bipartite graph in which every node has degree ≥ 1, has a perfect matching.

Now, we investigate the complexity of finding a perfect matching. The adjacency matrix of the graph G is an $n \times n$ 0-1 matrix $A = \{a_{ij}\}$ defined by:

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{1.17}$$

Observe that a perfect matching corresponds to a selection of n entries in the matrix with values 1 – i.e., exactly one selection for each row (and) each column. We define the *permanent* of the matrix as:

$$per(A) = \sum_{\phi \in S_n} a_{1\phi(1)} \times a_{2\phi(2)} \times ... \times a_{n\phi(n)} \tag{1.18}$$

where ϕ corresponds to a particular mapping of resources to tasks from the set of all possible mappings (given by S_n). Since the product $a_{1\phi(1)} \times a_{2\phi(2)} \times ... \times a_{n\phi(n)}$ corresponding to the mapping ϕ is 1 iff ϕ is a valid assignment, the permanent of the adjacency matrix provides a count of the number of perfect matchings in the bipartite graph G. The counting version of the perfect matching problem – i.e., evaluating permanent of a given graph – is a #PComplete problem [39]. The perfect matching problem is one of few problems whose counting version is known to be #PComplete, yet has a decision version known to be $\in$ P (via a reduction to the *max-flow problem*).

Theorem 4. *Bipartite Perfect Matching* (BPM) *problem* $\in$ P.

Proof. We define the *max-flow problem (MFP)* as follows: Given a bipartite graph $G = (U \cup V, E)$, with a source node s and sink node t such that each edge $(u,v) \in E$ has a capacity c_{uv}, what is the maximum flow routable from s to t?

We reduce the BPM to MFP by creating a new vertex s with outgoing edges to all vertices in U, and a new vertex t with incoming edges from all vertices in V. Finally, all edge capacities are set to 1. Then, there exists a perfect matching iff the max-flow is equal to $|U| = |V|$. □

From theorem 4, it is easy to see that valid assignments can be efficiently found using max-flow algorithms such as the Ford-Fulkerson[12] and Edmonds-Karp[11] algorithms. However, it should be clear that this may not help us find the optimal assignment (i.e., the assignment which maximizes some cost function). The Hungarian Algorithm [25] due to Harold Kuhn was the first polynomial time algorithm for optimally solving the personnel allocation problem (now known as the Linear Sum Assignment Problem ($LSAP$)), in 1955. However, most assignment problems often do not have exact algorithms that run in time polynomial in the size of the input. In fact, the generalized optimal assignment problem $\in$ NPHard (via a trivial reduction to the *0-1 Multiple Knapsack* problem [22]). Mathematically, the *Generalized Assignment Problem (GAP)* may be formulated as follows:

$$maximize: \sum_{i=1}^{n} \sum_{j=1}^{n} p_{ij} x_{ij} \tag{1.19}$$

$$subject\ to: \sum_{i=1}^{n} x_{ij} \leq 1, \forall j \in \{1,...,n\} \tag{1.20}$$

$$and: \sum_{j=1}^{n} c_{ij} x_{ij} \leq b_i, \forall i \in \{1,...,n\} \tag{1.21}$$

where

$$x_{ij} = \begin{cases} 1 & \text{if resource } i \text{ is assigned to task } j \\ 0 & \text{otherwise} \end{cases} \tag{1.22}$$

In words, the *GAP* is defined as follows: Given n resources $r_1, ..., r_n$, and n tasks $t_1, ..., t_n$. Each task (t_i) is associated with a budget b_i, and assigning resource r_j to task t_i results in an expense of c_{ij} and profit of p_{ij}. The goal is to find the assignment that results in the maximum profit.

1.3 Network Flows

As we will see in subsequent chapters, it is very useful to be able to model resource allocation problems as networks – i.e., graphs with edge costs, capacities, supply and demand nodes. Often, by relaxing some constraints it is possible to convert a hard allocation problem into a solvable instance of a network flow problem (particularly, linear network flows). These transformations are vital in designing efficient approximation techniques.

Formally, a *network* is defined as a directed graph $G = (V, E)$ such that

- There is a capacity c_e associated with each edge $e \in E$.
- There is a single source node (s) with no incoming edges and a sink node (t) with no outgoing edges.

We say that an *s-t flow* is a function that maps each edge to a real number, i.e., $f : E \rightarrow \mathsf{R}^+$, and the value $f(e)$ represents the amount of flow carried by the edge e [24]. Any flow f should satisfy the following conditions – (i) The flow along an edge should not exceed its capacity and (ii) The sum of flows entering any internal node should be equal to the sum of flows leaving it. We define the *value* of a flow ($v(f)$) as the amount of flow generated at the source – i.e., $v(f) = \sum_i f(e)$ where $e = (s, i)$.

The *Max Flow problem (MFP)* was first introduced by Tolstoi [38] in 1939 as part of an effort to regulate traffic flow in Russia. In general, the goal of the *MFP* is to find the maximum traffic that may be handled by a given network – i.e., given a network, what is the maximum possible flow value attainable?

Formally, the *MFP* may defined as follows: Given a bipartite graph $G = (U \cup V, E)$, with a source node s and sink node t such that each edge $(u, v) \in E$ has a capacity c_{uv}, what is the maximum flow routable from s to t? The problem is efficiently solvable – i.e., solvable in time that is polynomial in the input size.

The Ford-Fulkerson Algorithm

The *Ford-Fulkerson Algorithm (FFA)* [12] was the first method to compute the maximum flow in a given graph. Given a graph $G(V, E)$, with capacities $c_{uv}, \forall (u, v) \in E$, the algorithm outputs a flow $f(s, t)$ that is maximum. We denote the current flow between vertices u and v by $f(u, v)$. The *residual graph* of a network ($G_f(V, E_f)$) is defined as the network with capacity $c_f(u, v) = c_{uv} - f(u, v)$ and no flow.

The underlying principle of the algorithm is to repeatedly find paths from s to t with positive available capacity and assign a new flow along these paths in each iteration. Such paths are known as *augmenting paths*. In each step of the algorithm, flow conservation and capacity restrictions are maintained. Therefore, the final maximum flow is always legal. The algorithm itself is illustrated in 1.

When no more paths may be found, then there is no flow which may reach t from s and the algorithm terminates. While this termination is only when there are no more augmenting paths, there is no guarantee that this situation

Algorithm 1 The Ford-Fulkerson Algorithm for MFP

$f(u,v) \leftarrow 0$ forall $(u,v) \in E$
while there is a path p from s to t in G_f, where $\forall (u,v) \in p : c_f(u,v) > 0$ **do**
  $c_f(p) \leftarrow$ minimum edge capacity in p
  for each edge $(u,v) \in p$ **do**
    $f(u,v) \leftarrow f(u,v) + c_f(p)$
    $f(v,u) \leftarrow f(v,u) - c_f(p)$
  end for
end while

will ever arise when we use irrational flows. The run time of the algorithm, with integer flows and capacities, is $O(|E|f)$ where f is the maximum flow. There is a variation (due to Edmonds and Karp [11]) of the algorithm with a guaranteed termination time – $O(|V||E|^2)$ – i.e., independent of the maximum flow value.

While non-integer flows may cause problems and an infinite run time, the following theorem – *the Integral Flow Theorem* – guarantees that the algorithm terminates when flows are integers.

Theorem 5. *[The Integral Flow Theorem] If each edge in a flow network has integral capacity, then the maximum flow is an integer.*

Proof. (*Trivial*) The proof is by induction. It is easy to see that the base case is true – i.e., on the zero-th iteration of the *FFA*, the flow is integral since it is 0. We now assume that the flow is integral after the ith iteration. This implies that all the capacities in the residual network are also integral. Therefore, any augmenting path that is selected will have a integral flow. Since the addition of two integers is another integer, we have that the maximum flow after $i+1$ iterations of the FFA is also integral. □

The Edmonds-Karp Algorithm

The running time of the *FFA* depends on the selection of augmenting paths in the residual network. If poorly selected, given irrational capacities, it is possible for the algorithm to never terminate. To avoid this problem, the *Edmonds-Karp Algorithm (EKA)* is often used. The underlying principle of the EKA is identical to the FFA, with the only difference being the use of Depth-First search to compute the augmenting paths – i.e., the augmenting path is the shortest path from s to t in the residual network. The following analysis of the EKA is due to Rivest *et al* [34].

Theorem 6. *The total number of augmentations performed by the EKA is bounded by* $O(|V||E|)$

Proof. We define a critical edge as any edge in the residual network G_f that has residual capacity equal to the residual capacity of the augmenting path it belongs to – i.e., any edge $(u,v) \in p$ for which $c_f(p) = c_f(u,v)$, where p is the augmenting path. Clearly, every time flow is augmented, at least one critical edge is removed from the residual network. We will now show that each edge can become critical at most $\frac{|V|}{2} - 1$ times.

We define $\delta_f(u,v)$ be the shortest path between u and v in the residual network. Now, if u and v are connected by an edge, then, when (u,v) is a critical edge for the first time, we have: $\delta_f(s,v) = \delta_f(s,u) + 1$. When the flow is augmented, we no longer have (u,v) in the residual network. Further, it may not appear on another augmenting path until after the flow from u to v is decreased – i.e., when (v,u) appears in the augmenting path. If f' is such a

flow, then we have

$$\delta_{f'}(s,u) = \delta_{f'}(s,v) + 1 \tag{1.23}$$

$$\delta_{f'}(s,u) = \delta_{f}(s,u) + 2 \tag{1.24}$$

Therefore, between the time when (u,v) becomes critical for the second time, we have the distance from the source increased by two. Since we have no cycles, it is possible for the distance to only become at most $|V|-2$ before it is unreachable from s. Therefore, an edge (u,v) can only become critical up to $\frac{|V|-2}{2}$ times. Therefore, the number of augmentations cannot be more than $O(|V||E|)$. □

The running time of *EKA* is $O(|V||E|^2)$ – i.e., computing the new flows requires $O(|E|)$ for each iteration, and there cannot be more than $O(|V||E|)$ iterations (i.e., augmenting paths).

Another problem in network flows that has a significant impact on resource allocation problems is the *Min Cost Max Flow (MCMF)* problem. The problem was introduced by Klien in 1967 [23]. The goal is to find the cheapest legal flow among all maximum flows in a network with costs and capacities associated with each edge. This is a generalization of the *MFP* – i.e., the *MFP* is a special case of the *MCMF* problem where all edges have zero cost. The problem is solvable efficiently due to algorithms by Edmonds and Karp [11], and Goldberg and Tarjan [13], [14].

1.4 Chapter Summary

Chapter 1 provides the foundations critical to understanding the representation and approximation of optimal allocations. Section 1.1 contains a brief introduction to important concepts in *non-linear* and *convex programming*. Theorem 1 is a fundamental theorem which illustrates the benefit of optimizing over convex objective functions – i.e., finding a local minimum is sufficient for optimality – therefore making them efficiently optimizable using techniques such as subgradient projection [3]. The proof illustrated is due to Roberts [35]. The *generalized resource allocation* problem is shown to be $\in$ NPHard in Theorem 2. The result implies that in general, optimizing over inseparable functions taking only integer values is hard. The proof illustrated is a slight variation of the work of Ibaraki [20].

Section 1.2 introduces assignment problems as a subset of resource allocation problems. The concepts of *matchings* and *perfect matchings* are explained. The *Marriage Theorem* (Theorem 3) illustrates and proves the condition that is necessary and sufficient for the existence of a valid assignment (i.e., a perfect matching). The *bipartite perfect matching* problem is described and is proved to be efficiently solvable in Theorem 4. Finally, the *generalized assignment problem* is formulated. While it is not proved to be NPHard due to the triviality of the reduction, the proof may be found in [22].

Section 1.3 provides a very short introduction to network flows. The *max-flow problem* is formulated and the *Ford-Fulkerson*[12] and *Edmonds-Karp*[11] algorithms for finding the maximum flow in a network are described. Theorem 5 proves that integrality of flows is maintained when capacities are integral. This is critical for the working of the Ford-Fulkerson algorithm given integer capacities. The proof that the Edmonds-Karp algorithm terminates (even with irrational capacities) is illustrated in Theorem 6. The proof is due to Rivest and Leiserson [34].

Chapter 2

The Weapon-Target Allocation Problem

2.1 Introduction

The *Weapon-Target Allocation (WTA)* problem is a non-linear and stochastic military-offense related allocation problem that deals with how best to allocate missiles/weapons to enemy locations so as to inflict maximum damage. It is one of several problems in the field of command and control theory [2]. The problem may be stated as follows: Given a list of n targets $(t_1,...,t_n)$, a supply of m weapons $(w_1,...,w_m)$, and a *kill* matrix $P = \{p_{ij}\}$ with rational entries $\in (0,1)$ implying that the firing of weapon w_j fired at target t_i destroys it with probability p_{ij} $(= 1 - q_{ij})$. What allocation of weapons to targets causes the lowest surviving enemy targets? The allocation is done with the following assumptions:

- The kill probability of a certain weapon-target pair is independent of all other pairs. This assumption is not representative of real world engagements – i.e., it is possible that an enemy interception affects all missiles fired within some range of each other. However, it is required in order to isolate the allocation problem from the geometry of the problem.
- Damage surveys are completed only after all weapons have been fired (*single-round WTA*), or after a batch of weapons has been fired (*multi-round WTA*). The single round assumption is valid in short battles where there is only one opportunity to inflict damage to enemy targets, while the multi-round assumption holds in longer wars where weapons are supplied in batches or enemy targets are mobile (and may retreat).
- Only discrete allocations may be made – i.e., it is not possible to allocate a single weapon to multiple targets. However, multiple weapons may engage a single target.

2.1.1 Related Work

The Weapon-Target Allocation problem was first introduced in 1958 by Allan Manne [32] (attributed to Merrill Flood) as a non-linear program. In the same work, Manne proposed the first polynomial time approximation algorithm by making the assumption that kill probabilities are target dependent. Soon after, Den Broeder *et al.* [9] and Day [8] presented exact algorithms for several special cases of the WTA problem – when a single class of weapons are available and when a target may be attacked by only a single weapon. While significant, these works were not representative of real world battle scenarios. Lemus and David [28] provided yet another approximation algorithm

by treating the allocation matrix as a non-integral matrix (i.e., $x_{ij} \in (0,1)$ rather than $x_{ij} \in \{0,1\}$) and utilizing the Legrange multiplier method to find the global minimum of the resulting convex program. In 1986, Lloyd and Witsenhausen proved that long held suspicions about the complexity of the WTA problem were true by showing $WTA \in \mathsf{NPHard}$ [29]. Since then, several heuristics were proposed using concepts borrowed from generalized network flows [1], neural networks [40], and genetic algorithms[27] to find near optimal solutions efficiently. The multi-round WTA was largely ignored in early works, until Hosein and Athans presented approximations and analytical results in [16][17][18]. Hosein *et al.* showed that multi-stage WTA resulted in significantly better allocations than the single-stage version. Further, the concept of command and control centers were introduced in the enemy targets in [19]. A comprehensive survey of WTA based problems (with varying constraints) were presented by Murphey in [33]. Most recently, Dionne *et al.* and Rosenberger *et al.* presented exponential divide and conquer algorithms to compute optimal solutions to SWTA in [10] and [36], respectively.

2.2 Single Round Weapon-Target Allocation

The *Single Round WTA (SWTA)* may be stated as follows: Given a list of n targets $(t_1,...,t_n)$ and a supply of m weapons $(w_1,...,w_m)$ and a *kill* matrix $P = \{p_{ij}\}$ with rational entries $\in (0,1)$ implying that the firing of weapon w_j fired at target t_i destroys it with probability p_{ij} $(= 1 - q_{ij})$. What allocation of weapons to targets causes maximum enemy losses, assuming all allocated weapons are fired instantaneously?

2.2.1 Mathematical Formulation

The problem may be mathematically formulated as the following non-linear integer program.

$$minimize: \; F \; = \; \sum_{i=1}^{n} \left(\prod_{j=1}^{m} (q_{ij})^{x_{ij}} \right) \tag{2.1}$$

$$subject\ to: \; \sum_{i=1}^{n} x_{ij} = 1, \forall j \in \{1,...,m\} \tag{2.2}$$

where

$$x_{ij} = \begin{cases} 1 & \text{if weapon } w_j \text{ is assigned to target } t_i \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Equations (2.1) - (3.3) essentially state that the expected number of surviving targets (after *all* weapons have been fired) is to be minimized while allocating exactly one target to each weapon.

2.2.2 Complexity Results

Lemma 1. *Rational Product Dichotomy (RPD)* $\in \mathsf{NPHard}$.

Proof. Consider the following problem (RPD): Given a finite set of rational numbers $Q = \{q_1,...,q_n\}$ such that $q_i \in (0,1), \forall i \in \{1,...,n\}$, does there exist a subset K of Q (with complement K^c) such that: $\prod_{i \in K} q_i = \prod_{j \in K^c} q_j$? It is easy to see that $RPD \in \mathsf{NP}$ – i.e., it is possible to verify a given solution K in polynomial time.

We now show that *RPD* is NPHard by embedding in it an instance of the rational version of the subset product problem. The *Rational Subset Product (RSP)* problem is known to be NPHard [6] and may be stated as follows:

Given a finite set of rational numbers $Q = \{q_1, ..., q_n\}$ such that $q_i \in (0,1), \forall i \in \{1, ..., n\}$ and rational $r \in (0,1)$, does there exist a subset K of Q such that: $\prod_{i \in K} q_i = r$?

Given an instance $I_{RSP} =< Q, r >$ of *RSP*, we perform the following transformation into an instance $I_{RPD} =< Q' >$ of the *RPD* problem.

- Compute $P = \prod_{i=1}^{n} q_i$.
- If $P = r^2$. Then, set $Q' = Q$. Clearly a dichotomy is possible iff there is a subset of product r.
- If $P > r^2$. Then, set $q_{n+1} = \frac{r^2}{P}$. $P' = \prod_{i=1}^{n+1} q_i = r^2$, and $Q' = Q \cup \{q_{n+1}\}$. If there is a dichotomy, then there is always a subset with product r and vice-versa.
- If $P < r^2$. Then, set $q_{n+1} = \frac{P}{r^2}$. $P' = \prod_{i=1}^{n+1} q_i = r^2$, and $Q' = Q \cup \{q_{n+1}\}$. If there is a dichotomy, then there is always a subset with product r and vice-versa.

Clearly the above transformation is polynomial in size of the set Q. Therefore, any efficient algorithm for the *RPD* can be used to solve the *RSP* problem efficiently. Since *RSP* $\in$ NPHard, we have that *RPD* $\in$ NPHard.

□

Theorem 7. *Single-round WTA with two targets* $(SWTA_2) \in$ NPHard.

Proof. We define the decision version of $SWTA_2$ as follows: Given a value k, and a $2 \times m$ matrix $P = \{1 - p_{ij}\}$ with rational entries $\in (0,1)$. Is there a $2 \times m$ allocation matrix $X = \{x_{ij}\}$ such that $\sum_{i=1}^{2} \left(\prod_{j=1}^{m} (q_{ij})^{x_{ij}} \right) \leq k$ and $\sum_{i=1}^{2} x_{ij} = 1, \forall j \in \{1, ..., m\}$. Clearly, $SWTA_2 \in$ NP since it is easy to verify a given solution (i.e., X matrix) in polynomial time.

We now show that $SWTA_2 \in$ NPHard by embedding in it an instance of the *Rational Product Dichotomy (RPD)* problem (see lemma 1) such that an $SWTA_2$ solver returns *yes* iff the given instance of the *RPD* problem is also a *yes* instance. Given an instance $I_{RPD} =< Q >$ of the *RPD* problem, we make the following transformation into an instance $I_{SWTA_2} =< P, k >$ of the $SWTA_2$ problem. It is safe to assume that $\prod_{i \in Q} q_i = r^2$ (otherwise, the instance is trivially false). We set $p_{ij} = 1 - q_j$ for $i = 1, 2; \forall j \in \{1, ..., m\}$.

Any solution to the $SWTA_2$ problem has $x_{1j} + x_{2j} = 1, \forall j \in \{1, ..., m\}$. So for the given instance of $SWTA_2$, we have a yes instance iff $\prod_{j=1}^{m} q_j^{x_{1j}} + \prod_{j=1}^{m} q_j^{x_{2j}} \leq k$. To complete the reduction, we need to select the appropriate k. We do this by using the arithmetic-geometric means inequality [30] which states: For any list of n nonnegative real numbers $\{x_1, x_2, ..., x_n\}$, we have $\sum_{i=1}^{n} x_i \geq n \sqrt[n]{\prod_{i=1}^{n} x_i}$ and that equality holds if and only if $x_1 = x_2 = ... = x_n$.

Therefore, setting $k = 2\sqrt{\prod_{i=1}^{m} q_i} = 2r$ we have $\prod_{j=1}^{m} q_j^{x_{1j}} + \prod_{j=1}^{m} q_j^{x_{2j}} > k$ in all cases except when $\prod_{j=1}^{m} q_j^{x_{1j}} = \prod_{j=1}^{m} q_j^{x_{2j}}$ – i.e., when the *RPD* instance is true. When $k = 2r$, we have a *yes* instance of $SWTA_2$ iff there is a *yes* instance of the given *RPD* problem.

Clearly the above transformation is polynomial in size of the inputs. Therefore, any efficient algorithm for the $SWTA_2$ can be used to solve the *RPD* problem efficiently. Since *RPD* $\in$ NPHard, we have that $SWTA_2 \in$ NPHard.

□

The proof of hardness for *SWTA* with n targets ($SWTA_n$) is a trivial extension of the $SWTA_2$ proof illustrated for theorem 7. Further, the *SWTA* problem is a special case of the *Multi-round WTA (MWTA)* where the number of rounds is 1, therefore, *MWTA* is also NPHard.

2.2.3 Special Case Analysis

While theorem 7 shows that the *SWTA* problem is generally $\in$ NPHard, it only indicates that there does not exist an efficient algorithm to compute optimal allocations for *any* given instance of the problem. There are several special cases, however, which allow for efficient algorithms. In this section we review each of the known *easy* cases.

The Case of Equal Probabilities

When all p_{ij} have the same value $p(=1-q)$, then the *SWTA* problem reduces to the following non-linear program:

$$minimize: \ F \ = \ \sum_{i=1}^{n} (q^{a_i}) \tag{2.4}$$

$$subject\ to: \ \sum_{i=1}^{n} a_i = m \tag{2.5}$$

$$where: \ a_i = \sum_{j=1}^{m} x_{ij} \tag{2.6}$$

In words, the problem reduces to selecting non-negative integers $a_1, ..., a_n$ summing to m and minimizing F. This is easy to do in polynomial time as indicated by theorem 8.

Theorem 8. *F as defined in equation 2.4 is minimized iff m is divided as evenly as possible among the n a_i variables.*

Proof. We need to show that in any optimal solution, the values of a_i can take at most two consecutive integer values. If this does not hold, then there are indices i, j such that $a_i = k$ and $a_j = k+d$ where $(d > 1)$. From the following equation, it is easy to see that the contribution of the terms with these two indices is strictly reduced by transferring one unit from a_j to a_i.

$$(q^k + q^{k+d}) - (q^{k+1} + q^{k+d-1}) = q^k(1-q)(1-q^{d-1}) > 0 \tag{2.7}$$

Therefore, a_i, a_j as described can never be the optimal solution. This implies that the optimal solution is obtained only when a_i's are split as evenly as possible. □

Weapon Independent Probabilities

Recall the mathematical formulation of the *SWTA* problem (equations 2.1 - 3.3):

$$minimize: \ F \ = \ \sum_{i=1}^{n} \left(\prod_{j=1}^{m} (q_{ij})^{x_{ij}} \right)$$

$$subject\ to: \ \sum_{i=1}^{n} x_{ij} = 1, \forall j \in \{1, ..., m\}$$

where

$$x_{ij} = \begin{cases} 1 & \text{if weapon } w_j \text{ is assigned to target } t_i \\ 0 & \text{otherwise} \end{cases}$$

If the following additional constraint (eq. 2.8) is added to the above non-linear program, we have a special

efficiently solvable version of the *SWTA* problem – the *Weapon Independent SWTA (WI-SWTA)* problem.

$$P = \{p_{ij}\} = p_i, \forall j \in \{1,...,n\} \tag{2.8}$$

Equation 2.8 implies that each target has the same probability of being destroyed (regardless of which of the available weapons is used). The constraint may be valid in cases where only weapons of a single type are made available.

The *WI-SWTA* may be further simplified into the following non-linear program:

$$minimize: \ F \ = \ \sum_{i=1}^{n} \left(q_i^{x_i}\right) \tag{2.9}$$

$$subject\ to: \ \sum_{i=1}^{n} x_i = m \tag{2.10}$$

where

$$x_i = \sum_{j=1}^{m} x_{ij} \qquad and \qquad x_{ij} = \begin{cases} 1 & \text{if weapon } w_j \text{ is assigned to target } t_i \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

The function $F : \mathsf{Z}^+ \rightarrow R$ is convex and separable, making the problem significantly less difficult. We present an algorithm to find optimal solutions to *WI-SWTA* problems. The algorithm due to DenBroeder *et al.* [9] is known as the *Maximal Marginal Return (MMR)* algorithm and runs in $O(N + M \log N)$ time.

The Maximal Marginal Return (MMR) Algorithm: The MMR algorithm is a simple greedy approach that achieves optimality by assigning weapons sequentially to the targets for which the reduction in the objective cost is maximum. Algorithm 2 illustrates the algorithm in more detail. In each iteration the index k for which an increase in x_k by 1 causes the largest drop in the value of the objective function ($q_i^{x_i}$) is recorded. The value of x_k is then increased by 1, and the process is repeated until all weapons are allocated.

Algorithm 2 The MMR Algorithm for WI-SWTA

```
X ←< 0,0,...,0 >
for i = 1 → n do
    δ_i ← (q_i)^{x_i} − (q_i)^{x_i+1}
end for
for j = 1 → m do
    k : δ_k ← ARGMAX{δ_1, δ_2, ..., δ_n}
    x_k ← x_k + 1
    δ_k ← (q_k)^{x_k} − (q_k)^{x_i+k}
end for
```

Theorem 9. *The MMR Algorithm (Algorithm 2) achieves optimality for WI-SWTA.*

Proof. The following proof is by induction. When we have only one weapon to allocate – i.e., when $m = 1$, it is trivially true that the *MMR* algorithm achieves an optimal solution. We assume that the solution $X = (x_1, x_2, ..., x_n)$ returned by the *MMR* algorithm when $m = \tilde{m}$ is also optimal. Let k be the index of the element with the highest δ in vector X – i.e., $k : \delta_k^x = MAX(\delta_1^x, \delta_2^x, ..., \delta_n^x)$.

Let $m = \tilde{m} + 1$. Let the solution returned by the MMR algorithm be X^*. We know that $X^* = (x_1^*, x_2^*, ..., x_k^*, ..., x_n^*)$ = $(x_1, x_2, ..., x_k + 1, ..., x_n)$. We pick any other feasible solution $Z = (z_1, ..., z_n)$ such that $Z \neq X^*$. There must exist

some index j such that $z_j > x^*_j \geq x_j$. Let $\check{X} = (x_1, x_2, ..., x_j + 1, ..., x_n)$ and $\check{Z} = (z_1, z_2, ..., z_j - 1, ..., z_n)$. Now, the following equations are true (where $F(A) = f_1(a_1) + f_2(a_2) + ... + f_n(a_n)$ and $f_a(b_a) = q_a^{b_a}$):

$$F(Z) - F(\check{Z}) = f_j(z_j) - f_j(z_j - 1) \tag{2.12}$$

$$F(\check{X}) - F(X) = f_j(x_j + 1) - f_j(x_j) \tag{2.13}$$

Also, since X is optimal when $m = \tilde{m}$ and $z_j > x_j$, we have

$$F(X) \leq F(\check{Z}) \tag{2.14}$$

$$f_j(x_j) - f_j(x_j + 1) \geq f_j(z_j - 1) - f_j(z_j) \tag{2.15}$$

Using the above inequalities on the result of the difference between eq. 2.13 and 2.12 gives us $F(\check{X}) \leq F(Z)$. Since $\delta^x_k \geq \delta^x_j$, we also have $F(X^*) \leq F(\check{X})$. Therefore, X^* is an optimal solution when $m = \tilde{m} + 1$.

□

Initial ordering and computing of marginal returns requires $O(N)$ time. After each iteration, the newly updated marginal return has to be re-inserted in the appropriate position in the list – this can be done in $O(\log N)$ time. Since this is repeated upto M times, the overall complexity of the algorithm is $O(N + M \log N)$.

One Weapon per Target

Now, we consider the *SWTA* problem under the additional assumption that no target may be allocated more than a single weapon. We will show a reduction to the efficiently solvable transportation problem for the case where $m \leq n$. First, we make the following observations:

$$\prod_{j=1}^{m} (q_{ij})^{x_{ij}} = \prod_{j=1}^{m} (1 - p_{ij} x_{ij}) \tag{2.16}$$

$$\prod_{j=1}^{m} (1 - p_{ij} x_{ij}) = 1 - \sum_{j=1}^{m} p_{ij} x_{ij} \tag{2.17}$$

Equation 2.16 holds since $x_{ij} \in \{0, 1\}$, while equation 2.17 holds since $\sum_{j=1}^{m} x_{ij} \leq 1$. We can now formulate the *One Weapon Per Target SWTA (OW-SWTA)* as the following linear program.

$$maximize: \; F \; = \; \sum_{i=1}^{n} \sum_{j=1}^{m} (p_{ij} x_{ij}) \tag{2.18}$$

$$subject\ to: \; \sum_{i=1}^{n} x_{ij} = 1, \forall j \in \{1, ..., m\} \tag{2.19}$$

$$and \; : \sum_{j=1}^{m} x_{ij} \leq 1, \forall i \in \{1, ..., n\} \tag{2.20}$$

Now, consider the *transportation problem* defined as follows: Given a set of suppliers S and stores D, such that supplier i produces s_i goods, and store j has a demand for d_j goods. We assume that the cost of s_i supplying goods to d_j is given by c_{ij} per unit of goods. The goal is to find the minimum cost of supply such that all demands are

met and no supplies are exceeded. This may be formulated as the following linear program:

$$minimize: \ F \ = \ \sum_{i=1}^{n}\sum_{j=1}^{m}(x_{ij}c_{ij}) \tag{2.21}$$

$$subject\ to: \ \sum_{i=1}^{n} x_{ij} = d_j, \forall j \in \{1,...,m\} \tag{2.22}$$

$$and \ : \sum_{j=1}^{m} x_{ij} \leq s_i, \forall i \in \{1,...,n\} \tag{2.23}$$

It is easy to see that the *OW-SWTA* problem can be transformed into an instance of the transportation problem efficiently – i.e., by setting $c_{ij} = -p_{ij}$ and $d_j = s_i = 1$. Therefore, any algorithm that can solve the transportation problem efficiently may be used to solve the *OW-SWTA* problem in polynomial time (eg., min cost max flow algorithms [23]). Further, observe that when $m = n$, we have a special case of the bipartite matching problem.

2.2.4 Approximation Techniques

We have seen that there exist some special cases for which it is possible to efficiently solve the *SWTA* problem. However, for a vast majority of cases, this is not possible. It is therefore imperative to be able to efficiently approximate solutions and bound optimal solutions (to measure effectiveness of heuristics).

Integer Relaxation Based Approximation

The SWTA problem has several unique properties: (1) It is a non-linear integer program, (2) It models a stochastic process, (3) It requires scalable solutions – i.e., it needs to deal with allocations involving a large number of weapons and targets. It is the first property (non-linear integer program) that results in the problem being unsolvable (efficiently). Often, converting the problem to one where continuous allocations may be made (by relaxing the integer constraints) is useful for approximations. The resulting non-linear program then has a convex function which can be efficiently optimized using standard techniques. For the proof that the *SWTA* with the integer relaxation is indeed a convex function, we refer to the work of Hosein [16]. We refer to the resulting convex program (eq. 2.24 and 2.25) as the *Relaxed SWTA (R-SWTA)* problem.

$$minimize: \ F \ = \ \sum_{i=1}^{n}\prod_{j=1}^{m}(q_{ij})^{x_{ij}} \tag{2.24}$$

$$subject\ to: \ \sum_{i=1}^{n} x_{ij} = 1, \forall j \in \{1,...,m\} \tag{2.25}$$

The solution to the *R-SWTA* problem is the optimal solution if weapons could be fired at multiple targets – i.e., if we could fire fractional weapons. It is obvious that the solution to the *R-SWTA* problem is a lower bound on the optimal solution to the integer version of the problem (since the feasible solutions to the integer version are a subset of the feasible solutions of *R-SWTA*). One obvious technique for approximating solutions to the *SWTA* problem is by using standard techniques to solve the *R-SWTA* problem and then performing randomized rounding to achieve a near optimal solution to the integer counterpart.

Minimum Cost Flow Based Approximation

It is easy to see that the *SWTA* problem can be transformed into a maximization non-linear program – i.e., where the goal is to maximize the expected damage to targets. One way to obtain an approximation (or lower bound) is to upper bound this expected damage using network flow formulations. In this section we describe one such algorithm by Ahuja *et al*. [1]

Creating the Network Nodes: We create a network with three node layers (denoted by N_i). The first layer (N_1) contains a supply node i for each weapon type (we assume identical weapons are grouped together). The supply for each of these nodes is set to W_i – i.e., the number of identical weapons of type i. The second layer of nodes (N_2) contains nodes that correspond to the targets. We create m nodes ($j^1, j^2, ..., j^m$) for each target where j^i denotes the ith weapon striking the jth target. All the nodes in N_2 have a null supply and demand. Finally, we create a single sink node (t) in N_3 with demand equal to m (total number of weapons).

Drawing the Arcs: The network contains an arc (i, j^k) for each node $i \in N_1, j \in N_2$. This arc represents the allocation of a weapon of type i to the jth target as the kth weapon. An arc also connects each node $j \in N_2$ to the sink t. Therefore, we have an arc between each pair of nodes in N_1 and N_2, and also between each pair in N_2 and N_3. All arcs are assigned unit capacity.

A flow in this network is called a *contiguous flow* if it has the property that if $flow(i, j^k) = 1$, then $flow(i, j^l) = 1, \forall j \in \{1, ..., k-1\}$. It follows from this definition of a contiguous flow that there is a one-to-one correspondence between the feasible solutions of the given *SWTA* problem and the contiguous flows in the constructed network. However, finding this maximum flow is also a hard problem since the cost function is non-linear. To deal with this, we create a cost function for the network flow problem that is linear and overestimates the true non-linear costs.

Creating the Cost Function: Ideally, the cost function in the constructed network should reflect the amount of (expected) damage done by allocating a particular weapon to a given node – i.e., $c(i, j^1) = (p_{ij})$. However, consider the cost of $c(i, j^2)$ – the cost of the flow along this arc should reflect the damage done by allocating the ith weapon to the jth target as the second weapon. But this cost is clearly dependent on the first weapon that was allocated to it – i.e., the incoming arc on j^1. Therefore we cannot exactly determine the cost of the arcs $(i, j^2), ..., (i, j^m)$. In stead, we will compute the upper bound for each arc cost.

Suppose that the first $k-1$ weapons allocated to the jth target are of type $i_1, ..., i_{k-1}$, and the kth weapon is of type i_k. Then, the survival value (i.e., expected probability of no damage) of target j after the first $k-1$ weapons have been allocated is $q_{i_1} \times ... \times q_{i_{k-1}}$. This becomes $q_{i_1} \times ... \times q_{i_k}$ after the kth weapon has also been allocated. Hence, the cost of the arc (i, j^k) is $q_{i_1} \times ... \times q_{i_{k-1}} \times (1 - q_{i_k})$.

Now, let q_j^{max} be the maximum $q_{ij}, \forall i \in \{1, ..., m\}$. Then we can obtain an upper bound on $c(i, j^k)$ by replacing each q_{ij} by q_j^{max}. We get $c(i, j^k) = (q_j^{max})^{k-1}(1 - q_{ij})$. This gives us an upper bound on the amount of damage done by allocating the weapons to targets. Since $c(i, j^k) > c(i, j^{k+1})$, it is guaranteed that the maximum cost flow in the graph will be a contiguous one. Using any standard min cost flow algorithm, it is now possible to compute a near optimal allocation for the *SWTA* problem efficiently.

Neighborhood Search Based Approximation

Neighborhood search algorithms are a common method for solving problems with a large number of feasible solutions. The underlying principle is to start from a reasonably good feasible solution and gradually improve the objective functions value by replacing it with a better neighbor. The process is repeated until a locally optimal solution is obtained. In the case of convex functions, neighborhood search algorithms can be used to find globally

optimal solutions. Therefore, it is possible to apply this technique to solve the *R-SWTA* problem.

It is also possible to use neighborhood search techniques to obtain near optimal solutions to the *SWTA* problem (and other non-convex functions) itself without integer relaxation. However, in these cases, the quality of the locally optimal solution depends on the quality of the initial solution and the structure of the constructed neighborhood. In this section we describe one such technique (due to [1]) which builds on the generalized network constructed in section 2.2.4.

Starting Point Search Heuristic: First, the minimum cost flow for the network constructed in section 2.2.4 is computed. In the solution to this problem, there may be flow across arcs which have exact costs and arcs with approximate costs. The arcs with exact costs are *fixed* and removed – i.e., they are assumed to be part of the optimal solution and removed from the network. Now, it is possible to construct more exact cost arcs in the new graph (since some allocations are now known). The minimum cost flow in this new network is computed and the process is repeated until all arcs have exact costs. The procedure requires $O(m)$ iterations of the minimum cost flow algorithm and therefore finding the start point of the local search is efficient.

Neighborhood Structure: Let $S = \{S_1, ..., S_n\}$ denote a feasible solution to the *SWTA* problem where S_i denotes the set of weapons allocated to the ith target. A neighborhood is defined as the set of solutions which may be obtained from S by performing *multi-exchanges*. A multi-exchange is defined by some sequence of weapons (belonging to different subsets) – for eg., $i_1 i_2 ... i_r$ represents that weapon i_1 is given the target previously owned by weapon i_2 and i_2 is now assigned the target previously owned by i_3, etc. The multi-exchanges may be paths or cycles. While there are exponentially many neighbors for each solution and it is inefficient to enumerate them all, profitable multi-exchanges may be identified using *improvement graphs*.

Improvement Graphs: The improvement graph of a solution S, denoted by $G(S)$, contains nodes for each weapon and an arc between any two nodes with different targets – i.e., $(i, j) \in E(G(S))$ if i and j do not have the same targets. The arc has a cost equal to the change in objective value if the targets are swapped. If q_n is the expected survival cost of target j in the current solution, then the arc (i, j) is given the cost $c_{rl} = q_n(\frac{q_{in}}{q_{jn}} - 1)$. We call a directed cycle in $G(S)$ *subset disjoint* if each of the weapons in the cycle belong to different targets. Each of these disjoint subsets correspond to a valid multi-exchange in the *SWTA* problem [37].

The Neighborhood Search Algorithm: The algorithm is illustrated in 3. The improvement graph requires m nodes and can contain $O(m^2)$ arcs. Therefore the time to compute the cost of all the arcs is $O(m^2)$. Finding a cycle of length less than k in the graph requires $O(2^k)$ time, and computing the starting point requires $O(m^3 n^3)$ time. This gives us a running time of $O(m^2 2^k + m^3 n^3)$. k may be set to a pre-defined threshold – the value of k determines the quality of the outcome.

Algorithm 3 The SWTA Neighborhood Search Algorithm

```
S ← Starting-Point-Search-Heuristic(N)
G(S) ← Construct-Improvement-Graph(S)
while G(S) contains negative cost subset-disjoint cycle do
    W ← negative cost subset-disjoint cycle
    perform multi-exchange denoted by W
    update S
    update G(S)
end while
```

Table 2.1: Comparison of SWTA Approximation Heuristics

Scheme	Complexity	Worst Case	Max Opt Gap	Avg Opt Gap
Integer Relaxation	$O(min\{m^2, n^2\})$	20W, 40T	23.3%	12.1%
Minimum Cost Flow	$O(m^3 n^3)$	40W, 10T	42.4%	9.3%
Neighborhood Search	$O(m^3 n^3 + m^2 2^8)$	80W, 20T	2.3%	.8%

Experimental Results

Experimental results for the approximation heuristics described above were obtained by Ahuja *et al* in [1]. The goal was to identify which heuristic provided the tightest lower bound for the *SWTA* problem. To this end, 100 instances of the *SWTA* problem were generated with varying m and n values. For each instance of the *SWTA*, the optimal solution was computed and the gap between the solution by the heuristic and the optimal solution was reported.

In table 2.1, we report the worst case and average optimality gaps for each heuristic. As expected, the neighborhood search heuristic performed the best, however, this was at the expense of time for computing the approximate solution. In general, results from [1] indicate that the minimum cost flow heuristic performed fairly well in the case where $m \leq n$ (and failed when this was not the case), while integer relaxation did much better when $m > n$.

2.2.5 Branch and Bound Approaches

Branch and Bound algorithms [7] are a well known tool for finding exact solutions to difficult problems. While the branch and bound approach involves searching the entire solution space (i.e., set of feasible solutions) for the optimal solution, it does not require explicit enumeration – which in many cases (including *SWTA*) is impossible due to the exponential number of feasible solutions. Instead, such algorithms make use of *bounding functions* to decide which parts of the solution space are likely to contain the optimal solution.

At any point during the process of finding the optimal solution, the branch and bound algorithm status is defined by a pool of yet unexplored solutions, and the best currently known solution. The process is iterated until the pool of unexplored solutions is null. There are three principle components to each iteration: (i) Computing the bound for the current node using the bounding function, (ii) Expanding the current node, and (iii) Selection of the node to be processed next. If it is established that a node cannot possibly be the optimal solution, the entire set of solutions that contain the node as a part, are pruned and a new node is selected.

- *Bounding Strategy:* It is important to have a good bounding function in order to eliminate non-optimal branches from the solution tree quickly. Any of the approximation heuristics described in the previous section may be used as the bounding function. The study by Ahuja *et al.* [1], shows that the integer relaxation based approach produces tighter lower bounds than other approaches, however, requires more time. There is a trade-off between the quality of the bound and the time required to obtain it, and this must be taken into account while selecting the bounding function. While a low quality (and fast) bounding function may yield quicker approximations on each node, it may lead to the need for exploring more nodes than a higher quality (and slow) bounding function.

- *Branching Strategy:* One way suggested in [1] and [36] is to use a greedy approach such as the *MMR* algorithm – i.e., find the weapon-target combination with the best marginal return for the current node and set that as the node to be explored next. Ties may be broken arbitrarily.

Note, while the described method finds exact solutions without explicit enumeration, they are still not efficient (i.e., running in time that is polynomial in the size of the inputs), since it is possible that the entire solution space may end up being explored.

2.3 Multiple Round Weapon-Target Allocation

The *Multiple Round WTA (MWTA)* is a generalization of the *SWTA* which incorporates the concept of multiple time stages (battles). Here, the offense it allowed to observe the outcome of each battle and then assign weapons to targets for the next round. This is called the *shoot-look-shoot* strategy since the offense is allowed to alternate between firing weapons and surveying the damage inflicted.

We assume that in the initial stage the offense selects a subset of its weapons and allocates them to targets. These weapons are then fired simultaneously. The damage inflicted is then surveyed. Based on the information obtained, a list of surviving targets is generated and a subset of the remaining weapons are allocated and fired. The process is repeated until all weapons are exhausted, or targets are destroyed, or a threshold of rounds has been reached. The goal at each stage is to allocate weapons so that the expected number of surviving targets at the end of the final stage is minimized. It is clear that allocations are dependent solely on the outcome of the previous stage – i.e., the process is Markovian. In each stage, in order to achieve our goal, we need to obtain (i) the subset of weapons to be fired and (ii) a method to optimally allocate these weapons to targets. The *Principle of Optimality* applies in the sense that – in computing the optimal assignment for the current stage, one has to assume that optimal allocations will be made in all subsequent stages. If this is not followed, the optimal allocation will always improve the solution.

If the number of rounds remaining is known, then it is possible to re-state the problem as an *SWTA* problem at the start of each stage. Therefore, it is sufficient to find a method that provides optimal allocations for the first stage – all other stages can be solved (with the same method) by simply restating the problem as an *SWTA* problem.

2.3.1 Mathematical Formulation

In words, the *MWTA* problem may be stated as: Given a list of n targets $(t_1,...,t_n)$, a supply of m weapons $(w_1,...,w_m)$, an integer T, and T *kill* matrices $P(t) = \{p_{ij}(t)\}$ with rational entries $\in (0,1)$ implying that the firing of weapon w_j fired at target t_i in round t destroys it with probability $p_{ij}(t)$ $(= 1 - q_{ij}(t))$. What allocation of weapons to targets causes maximum enemy losses, assuming weapons may be fired in T batches and the evolution of the target list is observable? We define a decision indicator variable denoted by $x_{ij}(t)$.

$$x_{ij}(t) = \begin{cases} 1 & \text{if weapon } j \text{ is assigned to target } i \text{ in stage } t \\ 0 & \text{otherwise} \end{cases} \tag{2.26}$$

The *target state* of the system in stage t is defined as the set of targets which survive stages $1,...,t-1$. This is denoted by the n dimensional vector $U(t)$.

$$u_i(t) = \begin{cases} 1 & \text{if target } i \text{ survives stages 1,...,}t\text{-1} \\ 0 & \text{otherwise} \end{cases} \tag{2.27}$$

The *weapon state* is defined as the set of weapons available after stages $1,...,t-1$. This is denoted by the m

dimensional vector $W(t)$.

$$w_j(t) = \begin{cases} 1 & \text{if weapon } j \text{ was not used in stages 1,...,}t\text{-1} \\ 0 & \text{otherwise} \end{cases} \tag{2.28}$$

Given a first stage assignment $(x_{ij}(1))$, the target state $U(2)$ is a random vector. The probability that $u_i(2)$ is 1 is the probability that it survived the attack in stage 1. Similarly, the probability that $u_i(2)$ is 0 is the probability that it was destroyed by the attack in stage 1. The distribution of the variable $u_i(2)$ can therefore be given by:

$$Pr[u_i(2) = k] = k\prod_{j=1}^{m}(q_{ij}(1))^{x_{ij}(1)} + [1-k]\left(1 - \prod_{j=1}^{m}(q_{ij}(1))^{x_{ij}(1)}\right), \forall k \in \{0,1\}, \forall i \in \{1,...,n\} \tag{2.29}$$

We call this relation the *target state evolution* of the system. The weapon state also evolves with time. However, this evolution is deterministic and is given by $w_j(2) = 1 - \sum_{i=1}^{n} x_{ij}(1), \forall j \in \{1,...,m\}$ – i.e., the weapon is available in the subsequent stages iff it was not used in any of the previous stages. This relation is called the *weapon state evolution*. We denote the optimal solution to the $T-1$ stage problem with the initial target and weapon states U and W by $F_2^*(U,W)$. As a boundary condition, we set $F_{T+1}^*(U,W) = \sum_{i=1}^{n} u_i$ – i.e., we set the cost to the total value of targets that survived the final stage.

The *MWTA* problem can now be stated by the following non-linear program:

$$minimize:\ F_1 = \sum_{\omega \in \{0,1\}^n} Pr[U(2) = \omega] F_2^*(\omega, W(2)) \tag{2.30}$$

$$subject\ to:\ w_j(2) = 1 - \sum_{i=1}^{n} x_{ij}(1) \tag{2.31}$$

That is to say – the objective function to be minimized is the sum over all possible stage 2 target states of the probability that the target state is reached times the optimal cost given that state. The probability distribution of the target state $U(2)$ is known after the first round. This is done under the constraint of weapon evolution defined earlier. This problem is considerably more difficult than the *SWTA*. To see this, we consider the (seemingly simple) two stage problem. The number of possible weapon subsets that may be chosen in the first stage is 2^m. If a subset of size m_1 is selected, then there are n^{m_1} possible allocations. If there remain n_2 targets at the start of the second stage, then we have another $n_2^{m-m_1}$ allocations to be checked for optimality.

2.3.2 Analytical Results

It is known that the *MWTA* problem is NPHard since it is a generalized version of the *SWTA* problem. In this section, we report the analytical results due to Hosein in [16]. We do not present any new approximation techniques since the approximations presented for the *SWTA* problem are also applicable to *MWTA* problem at the start of each stage. Instead, we focus our attention on how to best decide which weapons are to be fired in each stage.

Stage Dependent Probabilities

Here we study the effect of having stage dependent probabilities on the optimal solution. We assume that these probabilities, given by $p(t)$, are independent of the target and weapons. We present several properties of the optimal solution in cases where the above assumption holds.

Theorem 10. *Consider a T stage MWTA with stage dependent kill probabilities. The optimal solution has the property that the weapons used in each stage are spread as evenly as possible among the surviving targets.*

Proof. The proof is by contradiction and is very similar to the proof for theorem 8. To simplify notation, we denote the kill probability for $t = 1$ by p (rather than $p(1)$). Similarly, the optimal number of weapons allocated to target t_i in the first stage is given by x_i. Let us assume that the allocation X is optimal, but not such that they are spread evenly – i.e., we assume $x_1 > x_2 + 1$. We denote the expected survival value for the optimal allocation by F^*. We have

$$F^* = q^{x_1+x_2}D_2 + [q_{x_1}(1-q^{x_2}) + q^{x_2}(1-q^{x_1})]D_1 + [(1-q^{x_1})(1-q^{x_2})]D_0 \tag{2.32}$$

where D_0 is the expected cost given that targets t_1 and t_2 are destroyed in stage 1, D_1 is the expected cost given one of the targets is destroyed in stage 1, and D_2 is the cost given that neither of the targets is destroyed in stage 1.

Now, we increase x_2 by 1 and reduce x_1 by 1 – i.e., we swap targets for one of the weapons allocated to the target t_1. Let the cost for this assignment be F. Then,

$$F = q^{x_1+x_2}D_2 + [q_{x_1-1}(1-q^{x_2+1}) + q^{x_2+1}(1-q^{x_1-1})]D_1 + [(1-q^{x_1-1})(1-q^{x_2+1})]D_0 \tag{2.33}$$

and we have $F^* - F > 0$. This contradicts our assumption that F^* is an optimal allocation. □

The above result simplifies the problem to be solved (to some degree). We now can use the number of weapons to be used in each stage (m_t) as the decision variable over which optimization is to be done. Once the optimal values of m_t are known, we simply spread these as evenly as possible in each stage to each target. We will see how to select values of m_t in the coming sections.

The next theorem deals with the case in which the number of weapons is less than the number of targets. As one might expect, if this is the case, *MWTA* based solutions do not perform better than *SWTA* based solutions. While it is not particularly enlightening, it reinforces the need to focus on how to split weapons into time stages when $m > n$.

Theorem 11. *If $m \leq n$, then the optimal solution is assign all weapons in the stage with the highest kill probabilities.*

Proof. We need to show that if $m \geq n$, and $p(t+1) > p(t)$ for $t \in \{1,...,T-1\}$, then $m' = (\sum_i m_i) = 0$. This is done using induction. The statement is trivially true when $T = 1$. We assume that it is also true for $T = t-1$ stages. Now, consider the $T = t$ stage *MWTA* problem. Suppose we assign m_1 weapons in the first stage, we have $m_1 \leq m \leq n$. By the induction assumption, all of the remaining weapons will be assigned in one of the following stages, the stage with the greatest kill probability. This means the problem is essentially reduced to a 2 stage problem. The expected survival value is given by:

$$F_1(m_1) = n - m + [(m-m_1)q(2)] + [m_1 q(1)] \tag{2.34}$$

where $q(1), q(2)$ are the survival probabilities of the targets in rounds 1 and 2, respectively. It is easy to see that when our assumption that $p(2) > p(1)$ holds, $F_1(m_1)$ is minimized when $m - m_1 = m$ – i.e., all the weapons are allocated in stage 2. □

However, it is only true that $m - m_1 \geq n$ if we have $p(t+1) \geq p(t)$ for $t \in \{1, ..., T-1\}$. The proof for this is also by induction and similar to the one above and is illustrated in [18].

The following observation concerns the case where the number of stages is very large. In this case, a single weapon is allocated to one of the surviving targets in each stage. This makes sense intuitively since if more than a single weapon is assigned to some target in a stage, and one of them destroys the target, the other weapon has essentially been wasted. Therefore, we have: *If* $T \to \infty$, $m > n$, *and* $p(t) = p$ *for* $t \in \{1, ..., T\}$ *then,* $m_1 = n$.

Finding the Optimal Number of First Stage Weapons for T = 2

We now attempt to find optimal values for the number of weapons to be allocated in each stage when $T = 2$ under the condition that probabilities are stage dependent. We introduce a new variable $k_i = \frac{m_i}{n}$. This is the ratio of weapons allocated per (initial) target in stage i. The expected cost of the T stage problem with n targets and for which $m_t = k_t N$ is given by $F_1(N, k_1)$. For the case where $T = 2$, we know that $k_2 = k - k_1$ (where $k = \frac{m}{n}$). Now, the optimization problem can be stated as follows:

$$minimize: \ F_2(\alpha, k_2) \tag{2.35}$$

$$subject\ to: \ k_1 \in [0, k] \tag{2.36}$$

Here, $\alpha = [1 - (k_1 - \lfloor k_1 \rfloor)p(1)](q(1))^{\lfloor k_1 \rfloor}$ – i.e., it is the expected fraction of targets which survive the first stage. The function $F_2(\alpha, k_2)$ can be given as:

$$F_2(\alpha, k_2) = [\alpha - p(2)(k_2 - \alpha \lfloor \frac{k_2}{\alpha} \rfloor]q(2)^{\lfloor \frac{k_2}{\alpha} \rfloor} \tag{2.37}$$

Notice that the integrality constraint makes the expression more complex and difficult to optimize. If we remove this requirement (allowing fractional assignments), we are left with a lower bound – i.e., $F_2(\alpha, k_2) \geq \alpha q(2)^{\frac{k_2}{\alpha}}$. This optimization problem (equations 2.38 - 2.39) is solvable efficiently and allows us to find a lower bound on the number of weapons to be allocated in each of the two stages.

$$minimize: \ \alpha q(2)^{\frac{k-k_1}{\alpha}} \tag{2.38}$$

$$subject\ to: \ k_1 \in [0, k] \tag{2.39}$$

If we have $\frac{k_2}{\alpha} = \lfloor \frac{k_2}{\alpha} \rfloor$, then the solution to this optimization problem is also a solution to the original – i.e., finding the number of weapons to be used in each round given that $T = 2$, $m > n$, and stage dependent probabilities is possible (efficiently) iff there exists a k_1 such that $\frac{k_2}{\alpha}$ is a positive integer.

Batched Arrival of Weapons

Another version of the *MWTA* problem is one where (identical) weapons arrive in batches (i.e., at the start of each stage). This problem is much simpler (but still $\in$ NPHard) than the above due to the fact that the optimal split of weapons for each stage is no longer required – i.e., we have no choice but to use all the weapons at hand – if the principle of optimality is to be obeyed. This problem reduces itself to a T-iteration *SWTA* problem where the list of targets and kill probabilities are updated after each iteration, but the weapons available are always static. This is also a special case of the standard *MWTA* problem where we assume an infinite supply of each weapon and only one of each may be used in each round.

2.4 Chapter Summary

Chapter 2 provides a detailed look at the current state-of-the-art in the analysis of the *weapon-target allocation* problem. In section 2.1, the problem is defined and assumptions are stated. This is followed by a brief description of work done on this problem to date, in subsection 2.1.1.

In section 2.2, the *single round WTA* problem is introduced. Subsection 2.2.1 formalizes the problem as a non-linear program. Subsection 2.2.2 analyzes the complexity of the *SWTA* problem. Theorem 7 (due to Lloyd and Witsenhausen [29]) proves that *SWTA* is an NPHard problem. It follows from this theorem that the *multi round WTA* is also NPHard. Subsection 2.2.3 examines special cases of the *SWTA* problem that are efficiently solvable. The optimality of solutions returned by the presented algorithms are proved in theorems 8 and 9. Theorem 9 is due to Hosein [16]. Subsection 2.2.4 lists several approximation heuristics and algorithms for finding near-optimal allocations. Several of the described heuristics are due to Ahuja *et al.* [1]. Finally, in subsection 2.2.5, a general approach to finding exact solutions using branch-and-bound algorithms is described.

In section 2.3, the *multi round WTA* problem is introduced. Subsection 2.3.1 formalizes the problem. The formalization presented is due to Murphey [33]. Subsection 2.3.2 presents an analysis of certain special cases of the *MWTA* problem. The complexity of the problem is shown by illustrating the hardness of even the simplest versions of the problem (for eg., the $T = 2$ *MWTA*). Approximation of the optimal split of weapons to be allocated is also described for the case where $T = 2$. This analysis is due to Athans *et al.* [17], [18] and Hosein [16]. Other approximations are not described since they are simple variations of the approximations illustrated for the *SWTA* problem.

Chapter 3

The Password Allocation Problem

3.1 Introduction

In this chapter we present the results of our brief investigation of a new RAP that deals with password (to account) allocations. The password allocation problem, in general, attempts to find an allocation of passwords to accounts in order to minimize risks associated with password compromise.

The *Basic Password Allocation Problem (BPA)* may be defined as follows: Given n accounts – $a_1, ..., a_n$, where each account (a_i) has an associated value (v_i) and compromise probability ($q_i = 1 - p_i$) . Is there a way to allocate k passwords (potentially, of varying strengths) to these n accounts (where, $k \leq n$) such that the Expected Gain (EG) is maximized (or, conversely – the Expected Loss (EL) is minimized)? This problem models many real world parallel job/resource allocation problems, and the results presented here are applicable in many domains including parallel process scheduling, job-machine allocations, etc.

In the case of the *PA* problem, accounts (a_i) may be of several types and assigning values (v_i) to them is not straight forward – for eg., some accounts hold sensitive data which may be assigned a monetary value (eg., bank accounts, retailer accounts, etc.), some may be gateways which may be used to gain access to other accounts (eg., email/IM accounts), and others may be fiscally worthless (eg., forum, club, alumni memberships), or invaluable (eg., blogs, social/professional media accounts). Further, different types of accounts/passwords may be more or less vulnerable to being compromised via password stealing attacks. Therefore, we make the following assumptions around which we formalize our model:

- If account j is allocated to password i, there is a given probability q_i (if passwords are of equal strength) or q_{ij} (otherwise) that this password will be compromised (due to an attack on j). This probability is independent of other probabilities and events.
- If a password i is compromised due to an attack on any account allocated to it, it is assumed that all the accounts allocated to it are compromised.
- Every account may be associated with a monetary value v_i in a way that is independent of the values of other accounts.
- Gain/Loss assessment is not performed until all accounts have been allocated.

Ideally, we would like to allocate different passwords to each account – so the compromise of a single account would not result in the compromise of multiple accounts. However, typically, users have a fixed set of a few (k) passwords, which are to be allocated to many (n) accounts. In such situations, we would like to be able to allocate passwords to accounts so as to minimize the expected monetary loss over a period of time.

3.2 Mathematical Formulation

The *BPA* problem may be mathematically formulated as the following non-linear integer program.

$$maximize: \ F = \sum_{i=1}^{k} \left(\prod_{j=1}^{n} ((p_j)^{x_{ij}}) \sum_{j=1}^{n} (x_{ij} v_j) \right) \tag{3.1}$$

$$subject\ to: \ \sum_{i=1}^{k} x_{ij} = 1, \ \forall j \in \{1, ..., n\} \tag{3.2}$$

where

$$x_{ij} = \begin{cases} 1 & \text{if account } j \text{ is assigned password } i \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

We call the objective function F (in eq. 3.1) the expected gain (EG) of an allocation. This is the expected surviving value of accounts after all allocations have been made. The constraint in this definition enforces that each account is assigned exactly one password.

An equivalent problem is the minimization version given by

$$minimize: \ F = \sum_{i=1}^{k} \left((1 - \prod_{j=1}^{n} p_j^{x_{ij}}) \sum_{j=1}^{n} (x_{ij} v_j) \right) \tag{3.4}$$

$$subject\ to: \ \sum_{i=1}^{k} x_{ij} = 1, \ \forall j \in \{1, ..., n\} \tag{3.5}$$

The objective function F to be minimized in 3.4 is called the expected loss (EL) of an allocation.

Proposition 1. *The problems of finding assignments which maximize expected gain and minimize expected loss are equivalent.*

Proof. The expected gain (EG) and expected loss (EL) of an assignment of passwords to accounts are given by

$$EG = \sum_{i=1}^{k} \left(\prod_{j=1}^{n} (p_j^{x_{ij}}) \sum_{j=1}^{n} (x_{ij} v_j) \right)$$

$$EL = \sum_{i=1}^{k} \left((1 - \prod_{j=1}^{n} p_j^{x_{ij}}) \sum_{j=1}^{n} (x_{ij} v_j) \right)$$

$$\implies EG + EL = \sum_{i=1}^{k} \left(\sum_{j=1}^{n} (x_{ij} v_j) \right) = \sum_{j=1}^{n} v_j.$$

Therefore, an assignment that maximizes the expected gain, also minimizes the expected loss and vice-versa. □

3.3 Complexity Results

Proposition 2. *If $EG(S_1)+EG(S_2) \geq \sqrt[2]{P} \times V$, where $V = \sum_{i=1}^{n} v_i$, $P = \prod_{i=1}^{n} p_i$, and the elements in S_1 and S_2 exactly cover all elements in $\{1,...,n\}$, then either $V_{S_1} = V_{S_2} = \frac{V}{2}$, or $V_{S_2} < V_{S_1}$ and $\frac{P_{S_2}}{P_{S_1}} \leq 1$, or $V_{S_2} < V_{S_1}$ and $(\frac{P_{S_2}}{P_{S_1}}) \geq (\frac{V_{S_1}}{V_{S_2}})^2$.*

Proof. Solving the normalized system of linear equations given by:

$$P_{S_1}V_{S_1} + P_{S_2}V_{S_2} \geq \sqrt[2]{P_{S_1}P_{S_2}} \tag{3.6}$$

$$V_{S_1} + V_{S_2} = 1 \tag{3.7}$$

yields the following solutions:

$$V_{S_2} < V_{S_1},\ (\frac{P_{S_2}}{P_{S_1}}) \geq (\frac{V_{S_1}}{V_{S_2}})^2 \tag{3.8}$$

$$V_{S_2} < V_{S_1},\ (\frac{P_{S_2}}{P_{S_1}}) \leq 1 \tag{3.9}$$

$$V_{S_1} = V_{S_2} = \frac{1}{2} \tag{3.10}$$

□

Note that the solution given by equation 3.10 is independent of the values of P_{S_1} and P_{S_2}. This will be a critical observation in the following proof of hardness (Theorem 12). Equation 3.10 holds due to the parameterized arithmetic-geometric mean inequality due to Cauchy (1821) [5].

Now, we define the Integer Subset Dichotomy problem (*SD*) as follows: Given a set of n positive integers $Q = \{q_1,...,q_n\}$, does there exist a partition of Q into two covering sets Q_1 and Q_2, such that $\sum_{i \in Q_1} q_i = \sum_{j \in Q_2} q_j$?

Lemma 2. *Integer Subset Dichotomy (SD)* $\in$ NPHard.

The proof (a reduction via subset sum) is omitted due to its triviality.

Theorem 12. *The Basic Password Allocation problem with two passwords (BPA_2) is NP-complete.*

Proof. We can formulate a decision version of the BPA_2 problem as follows: Given a set of n integers $V = \{v_1,...,v_n\}$, and rationals $P = \{p_1,...,p_n\} \in (0,1)$, does there exist a partition of $N = \{1,...,n\}$ into two covering sets S_1, S_2 such that $EG_{S_1} + EG_{S_2} \geq r$?

We first show that BPA_2 is in NP. Given a set of n integers $V = \{v_1,...,v_n\}$, and rationals $P = \{p_1,...,p_n\} \in (0,1)$, and a target r, a certificate that there is a solution would be the two subsets S_1 and S_2 that have *EGs* that sum to a value greater than r. In polynomial time, we can compute the EG of each subset and verify that their sum $\geq r$.

We now show that $BPA_2 \in$ NPHard by embedding in it an instance of the *Integer Subset Dichotomy (SD)* problem (a simple transformation of Subset Sum) such that a BPA_2 solver returns *yes* iff the given instance of the *SD* problem is also a *yes* instance. Formally, we define the *SD* problem as follows: Given a set of n positive integers $Q = \{q_1,...,q_n\}$, does there exist a partition of Q into two covering sets Q_1 and Q_2, such that $\sum_{i \in Q_1} q_i = \sum_{j \in Q_2} q_j$?

Given an instance $I_{SD} =< Q >$ of the *SD* problem, we make the following transformation into an instance $I_{BPA_2} =< P,V,r >$ of the BPA_2 problem. Let μ_1 be the smallest element in Q and let μ_2 be the sum of all other elements in Q. Now, let x be some rational number $\in (0,1)$ such that $x^2 < (\frac{\mu_1}{\mu_2})^2$. Now, we set $V \in I_{BPA_2} = Q \in I_{SD}$, the compromise probabilities P in the instance of BPA_2 as follows: $p_i = x^{v_i}$, and $r = \sqrt{\prod_{i=1}^{n} p_i} \times \sum_{i=1}^{n} v_i$. Note that this transformation is valid since it requires time polynomial in the size of the inputs.

We can make the following observation from this transformation:

- (wlog) When $\frac{V_2}{V_1} < 1$, we always have $\frac{P_2}{P_1} \in (1, \frac{V_1}{V_2}^2)$.

In conjunction with proposition 2, it is implied that the BPA_2 solver returns *yes* iff the given instance of *SD* is also a a *yes* instance. Therefore, any instance of *SD* can be solved in polynomial time given a BPA_2 solver. Since $SD \in \mathsf{NPHard}$, we have that $BPA_2 \in \mathsf{NPHard}$.

□

The above proof of hardness is easily extended to the case where $k > 2$ and the case where passwords have different strengths – i.e., BPA_k and *the Generalized PA problem* $\in \mathsf{NPHard}$.

3.4 Analytical Results

Examination of several simplified cases of the *PA* problem reveal some of the complications to be expected in more general cases. To this end, we first demonstrate how optimal allocations are done in simple cases.

The Case of $n = k$

Proposition 3. *When $n = k$, an optimal assignment always allocates exactly one account to each password.*

Proof. (trivial) Since $0 \leq p_j \leq 1$, and $v_j \geq 1$, $\forall j \in \{1,...,n\}$ it always holds that

$$(v_s + v_t)(p_s \times p_t) \leq (v_s \times p_s) + (v_t \times p_t), \forall s,t \in \{1,...,n\} \tag{3.11}$$

The above inequality indicates that the expected gain is maximized when an assignment has only one account per password. □

Further, from proposition 3, it is also clear that any optimal allocator for any n and k (where $n \geq k$), will never generate an allocation containing a password with no accounts allocated to it.

The Case of $n = k + 1$

In this situation, the greedy approach shown in algorithm 4, finds an optimal assignment in $O(n)$ time. Note: This is a simple extension of proposition 3, and a proof that this is optimal follows directly.

Algorithm 4 Calculate Optimal Assignments for $n = k + 1$

```
for j = 1 → n do
    G_j ← p_j × v_j
end for
index1 ← find-remove-return-smallest-number-index(G_1, ..., G_n)
index2 ← find-remove-return-smallest-number-index(G_1, ..., G_{n-1})
x_{n-1,index1} ← 1
x_{n-1,index2} ← 1
for j = 1 → n − 2 do
    x_{j,j} ← 1
end for
```

The Case of $n \geq k+2$

While the case of $n = k+1$ was simple to deal with, we will see that things get exponentially more complicated as n increases (with respect to k). To understand why, consider the simple case of $n = 4$, and $k = 2$ – i.e., we want to allocate 4 accounts (A_1, A_2, A_3, A_4) to 2 passwords (PW_1, PW_2).

Assuming (w.l.o.g) that $p_4v_4 \geq p_3v_3 \geq p_2v_2 \geq p_1v_1$, there are four possible allocations, each of which are optimal under different conditions. For eg., $\{A_4\} \in PW_1$, $\{A_1, A_2, A_3\} \in PW_2$ is optimal only when $(1-p_3)(p_4v_4 - 2p_1p_2v_2) + p_3v_3(p_1p_2 - p_4) \geq 0$, and $(p_2 - p_3)(p_1v_1 - p_4v_4) + (p_1 - p_4)(p_2v_2 - p_3v_3) \geq 0$.

The Case of Identical Accounts

When $p_1 = p_2 = ... = p_n = p$ and $v_1 = v_2 = ... = v_n$=v, the *PA* problem reduces to choosing non-negative integers a_i for $i = 1, ..., k$ to maximize:

$$\sum_{i=1}^{k} a_i \times p^{a_i} \tag{3.12}$$

under the constraint that $\sum_{i=1}^{k} a_i = n$.

In this case, the optimal assignment shows a gradual progression from the most skewed assignment with every password allocated at least one account (i.e., $n-k+1$ accounts allocated to a single password, and each of the other $k-1$ passwords get exactly 1 account), towards a uniform assignment as p increases.

Representing Allocations as Polynomials: From equation 3.12, it is clear that any allocation can be represented as a polynomial with degree in the range of $\left[n-k+1, \lceil \frac{n}{k} \rceil\right]$, moving from the left end of this range towards the right, as p increases. For eg., with $n = 100$, and $k = 2$, when $p \to 0$, the optimal allocation can be represented by the polynomial $99p^{99} + p$, and when $p \to 1$, it can be represented by $100p^{50}$.

There are exactly $\lceil \frac{\binom{n-1}{k-1}}{k!} \rceil$ such polynomials (i.e., as many as the number of unique ways to sum to n with k elements). We label polynomials by their *skewedness* – i.e., EQ_i is the i^{th} most skewed allocation, $i \in \{1, ..., \lceil \frac{\binom{n-1}{k-1}}{k!} \rceil\}$. Generating these polynomials is a constant time operation, and all of them may be generated in $O(\frac{\binom{n-1}{k-1}}{k!}) \approx O(\frac{n^k}{2\pi k(k/e)^{2k}})$ time.

A $O(n^{k+6})$ Algorithm: Once we have a set of ordered and labeled polynomials, the following algorithm may be used to find an optimal allocation for a given p', in $O(n^6 \times \frac{\binom{n-1}{k-1}}{k!}) \approx O(n^6 \times (\frac{n}{k^2})^k)$.

Algorithm 5 Linear-Compute-Optimal(EQ_i, p')

```
Solve for p in EQ_i - EQ_{i-1} = 0
if p ≥ p' then
    return EQ_i
else
    Linear-Compute-Optimal(EQ_{i+1}, p')
end if
```

Exponential speed-up may easily be achieved by using binary range searching, rather than linear range searching in order to achieve a solution in $O(n^6 \times \log(\frac{n}{k^2})^k)$. Observe however, that the bottleneck in this algorithm does not lie in finding the optimal range, but in generating the polynomials that represent different ranges. Note, the n^6 term is representative of the time required to solve high degree polynomial equations.

A $O(nk)$ Algorithm The $O(nk)$ algorithm is a simple greedy approach that achieves optimality by assigning accounts sequentially to the passwords for which the increase in the objective cost is maximum. Algorithm 6 illustrates the algorithm in more detail. In each iteration the index k for which an increase in x_k by 1 causes the largest increase in the value of the objective function ($vx_i p^{x_i}$) is recorded. The value of x_k is then increased by 1, and the process is repeated until all accounts are allocated passwords.

Algorithm 6 A $O(nk)$ Algorithm for Identical Accounts

```
for i = 1 → n do
    for j = 1 → k do
        EG_old ← EG_j
        S_j ← S_j + a_i
        EG_j ← compute-EG(S_j)
        δ_j ← EG_j − EG_old
        S_j ← S_j − a_i
    end for
    x ← get-max-index(δ_1, ..., δ_k)
    S_j ← a_x
end for
```

The proof that the procedure achieves optimality when identical accounts are to be allocated follows from theorem 9.

3.5 Approximation Techniques

The following approximation techniques have so far been considered for finding near optimal solutions to the *BPA* problem. Note that not all techniques have so far been analyzed and investigated, and the following are simply suggested techniques.

3.5.1 Integer Relaxation Based Approximation

As we saw with the *SWTA* problem, converting the problem to one where continuous allocations may be made (by relaxing the integer constraints) is useful for approximations. The resulting non-linear program then has a convex function which can be efficiently optimized using standard techniques. We refer to the resulting convex program as the *Relaxed Password Allocation (R-PA)* problem. The *RPA* problem returns an optimal solution if it were possible to assign fractional passwords to accounts. While it is not clear exactly what this means, it is a lower bound on the solution to the *BPA* problem.

Further, when randomized rounding is performed on the optimal solution to the *RPA* problem, it becomes a useful and efficient starting point for more complex heuristics such as those using neighborhood based search techniques.

3.5.2 Best EG First Greedy Approximation

The best EG first algorithm works on the *greedy* principle that an account is more likely to be allocated to the password whose EG increases the most (or) decreases the least after the allocation. The accounts are ordered in a pre-determined manner (eg., by $p_i \times v_i$), and each account is allocated to the password whose EG increases the

most (or) decreases the least after the allocation (every password has an initial EG of zero). The time required for allocation with the Best EG First heuristic is $O(nk+n\log n)$. The post ordering of accounts procedure is illustrated in Algorithm 6. It is important to keep in mind that the ordering of accounts being allocated significantly affects the outcome and effectiveness of the algorithm. In future work, the effectiveness of different orderings will be analyzed.

3.5.3 K-Means Clustering

The K-Means clustering algorithm obtains a partition of d dimensional data points into voronoi cells with minimum within-cluster sum of difference from cluster means. While the problem of partitioning data into k clusters with minimum within-cluster sum of squares itself is known to be NP-complete, when the number of dimensions and clusters are fixed, simple algorithms can be achieved to get optimal results in $O(n^{dk+1}\log n)$ time, where d is the number of dimensions in each data point and k is the number of clusters. In our experiments, the number of dimensions $d = 2$ – the value and the probability of each account.

Algorithm 7 K-Means(P,V,k)

$C_i \leftarrow random_i(\forall i \in \{1,...,n\})$
for $i \in \{1,...,n\}$ **do**
  $m(i) = argmin_{k\in\{1,...,n\}} distance(i, C_k)$
end for
if m has changed **then**
  for $i \in \{1,...,k\}$ **do**
    C_i = **mean**($members(i)$)
  end for
  for $i \in \{1,...,n\}$ **do**
    $m(i) = argmin_{k\in\{1,...,n\}} distance(i, C_k)$
  end for
end if
return *members*

In the above algorithm, C_k denotes the centroid of the kth cluster, $m(i)$ denotes the cluster membership of the ith account, and $members(i)$ denotes the accounts allocated to the ith cluster.

3.6 Chapter Summary

Chapter 3 describes current work on the analysis of a new probabilistic allocation problem know as the *password allocation* problem. In section 3.1, the problem is described and its assumptions are stated.

Section 3.2 provides the formalization required to represent the problem as a non-linear program. The *basic PA* problem is defined by all passwords being of equal strength. The *generalized PA* problem is defined by allocating accounts to passwords of different strengths. The complexity of the problem is explored in Section 3.3. It is shown via a reduction to the *subset dichotomy* problem that the PA problem is NPHard.

Several special cases of the problem are explored in section 3.4 along with efficient algorithms for the cases that are known to be solvable in polynomial time. Finally, in section 3.5, several heuristics are described for generating near-optimal solutions. The content of this chapter is a work in progress, and results are forthcoming.

Chapter 4

Conclusions

In this report we provided an analysis of a class of probabilistic resource allocation problems. In particular, the Weapon-Target Allocation and the Passwords Allocation problems were studied. The analysis is applicable to many other resource allocation problems where there exists a possibility of failure – i.e., it is not guaranteed that a resource will be able to finish the allocated task.

First, in chapter 1 a brief introduction to vital concepts of non-linear programming and approximation techniques was given. In particular, the hardness of convex programming, generalized resource allocations, generalized assignments, and maximum flow problems was discussed.

In chapter 2, a detailed analysis of the single round and multi round WTA was provided. In particular, the problems were shown to be NPHard. For the single round WTA, several efficiently solvable special cases were identified. However, it was observed for the multi round WTA, that only trivial (unrealistic) cases were solvable. The problem of deciding which weapons were to be allocated at the start of each round was by itself shown to be hard. Several approximation heuristics were described for the single round WTA. Note: the provided heuristics are valid for approximating solutions to the multiple round version – once the weapons to be used in each round are known.

Chapter 3 described our current work on the analysis of a new probabilistic allocation problem know as the *password allocation* problem. The password allocation problem, in general, attempts to find an allocation of passwords to accounts in order to minimize risks associated with password compromise. The main contribution of this chapter was a study of the complexity of the *Basic PA* problem. In particular, the *PA* problem and its generalized variant were shown to be NP-complete. Further, several heuristics for finding approximate solutions were identified.

4.1 Future Work

Much of our future work lies in expanding the results presented in chapter 3. Our future work lies in two main categories.

- **Approximation Techniques and Algorithms:** It is clear that several approximations for the WTA problem (eg., the local search algorithm) may be easily adapted to fit the password allocation problem. As part of future work, we plan on adapting such algorithms and validating their effectiveness. However, we are more interested in being able to prove the existence (or non-existence) of constant-factor approximation

algorithms. We believe that heuristic approaches such as those described in chapters 2 and 3 cannot be reliable enough for solving real world instances of critical allocation problems. Due to the similar nature of the WTA problem and the PA problem, we hope that our results will be applicable for both problems.

- **Online Version of the *PA* Problem:** The second part of our future work entails extending the *PA* problem into a dynamic online problem. For our analysis on the *PA* problem to be truly useful in the real world, it is critical that we account for changing conditions – for eg., evolving accounts and password strengths, evolving probabilities of compromise, etc. This is also particularly useful for parallel process scheduling – where processes arrive in regular periods of time and older processes simply expire. It is clear to us that the problem is not efficiently solvable, and the multi round WTA gives us good reason to believe that analysis may yield few results. However, it is a logical extension of our work and must be attempted.

Bibliography

[1] R. K. Ahuja, A. Kumar, K. C. Jha, and J. B. Orlin. Exact and heuristic algorithms for the weapon-target assignment problem. *Oper. Res.*, 55(6):1136–1146, Nov. 2007.

[2] M. Athans. Command and control (c2) theory: A challenge to control science. *Automatic Control, IEEE Transactions on*, 32(4):286–293, 1987.

[3] D. Bertsekas. *Nonlinear programming*. Athena Scientific, 1999.

[4] G. Bitran and A. Hax. Disaggregation and resource allocation using convex knapsack problems. *Management Science*, 27(4), 1981.

[5] A. Cauchy. *Cours d'analyse de l'Ecole royale polytechnique: Analyse algébrique*. Debure, 1821.

[6] M. Cieliebak, S. Eidenbenz, A. T. Pagourtzis, and K. Schlude. On the complexity of variations of equal sum subsets. *Nordic J. of Computing*, 14(3):151–172, Sept. 2008.

[7] J. Clausen. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen*, 1999.

[8] R. Day. Allocating weapons to target complexes by means of nonlinear programming. *Operations Research*, pages 992–1013, 1966.

[9] G. den Broeder Jr, R. Ellison, and L. Emerling. On optimum target assignments. *Operations Research*, pages 322–326, 1959.

[10] D. Dionne, E. Pogossian, A. Grigoryan, J. Couture, and E. Shahbazian. An optimal sequential optimization approach in application to dynamic weapon allocation in naval warfare. In *Information Fusion, 2008 11th International Conference on*, pages 1 –6, 2008.

[11] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, Apr. 1972.

[12] L. Ford and D. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.

[13] A. Goldberg and R. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM (JACM)*, 36(4):873–886, 1989.

[14] A. Goldberg and R. Tarjan. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research*, pages 430–466, 1990.

[15] P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 10(1):26–30, 1935.

[16] P. Hosein. A class of dynamic nonlinear resource allocation problems. Technical report, DTIC Document, 1989.

[17] P. Hosein and M. Athans. An asymptotic result for the multi-stage weapon-target allocation problem. In *Decision and Control, 1990., Proceedings of the 29th IEEE Conference on*, pages 240–245. IEEE, 1990.

[18] P. Hosein and M. Athans. Some analytical results for the dynamic weapon-target allocation problem. Technical report, DTIC Document, 1990.

[19] P. Hosein, J. Walton, and M. Athans. Dynamic weapon-target assignment problems with vulnerable c2 nodes. Technical report, DTIC Document, 1988.

[20] T. Ibaraki. Solving mathematical programming problems with fractional objective functions. *Generalized Concavity in Optimization and Economics*, 1981.

[21] T. Ibaraki and N. Katoh. *Resource allocation problems: algorithmic approaches*. MIT Press, 1988.

[22] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer Verlag, 2004.

[23] M. Klein. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Management Science*, 14(3):pp. 205–220, 1967.

[24] J. Kleinberg. *Algorithm Design*. Pearson Education, 2006.

[25] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[26] H. W. Kuhn and A. W. Tucker. Non linear programming. *Proceedings of Second Berkeley Symposium on Mathematics, Statistics, and Probability*, pages 481–492, 1951.

[27] Z. Lee, S. Su, and C. Lee. Efficiently solving general weapon-target assignment problem by genetic algorithms with greedy eugenics. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 33(1):113–121, 2003.

[28] F. Lemus and K. David. An optimum allocation of different weapons to a target complex. *Operations Research*, pages 787–794, 1963.

[29] S. Lloyd and H. Witsenhausen. Weapon-target allocation is np complete. *Proceedings of the Summer Computer Simulation Conference*, pages 1054–1059, 1986.

[30] A. Lohwater. *Introduction to Inequalities*. Marjorie Lohwater, 1982.

[31] H. Luss and S. K. Gupta. Allocation of effort resources among competing activities. *Operations Res.*, 23(2):360–366, 1975.

[32] A. Manne. A target-assignment problem. *Operations Research*, pages 346–351, 1958.

[33] R. A. Murphey. Target based weapon target assignment problems. In *Nonlinear assignment problems: algorithms and applications*, volume 7. Springer, 2000.

[34] R. Rivest and C. Leiserson. *Introduction to Algorithms*. McGraw-Hill, Inc., 1990.

[35] A. Roberts and D. Varberg. *Convex functions*, volume 57. Academic Press, 1973.

[36] J. Rosenberger, H. Hwang, R. Pallerla, A. Yucel, R. Wilson, and E. Brungardt. The generalized weapon target assignment problem. Technical report, DTIC Document, 2005.

[37] P. Thompson, J. Orlin, and M. I. of Technology. Operations Research Center. *The Theory of Cyclic Transfers*. Working paper OR. Operations Research Center, Massachusetts Institute of Technology, 1989.

[38] A. Tolstoı. Methods of removing irrational transportation in planning. *Sotsialisticheskiı Transport*, 9:28–51, 1939.

[39] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[40] E. Wacholder. A neural network-based optimization algorithm for the weapon-target assignment problem. Technical report, DTIC Document, 1989.