11-9-2004

An Event Driven Single Game Solution For Resource Allocation In A Multi-Crisis Environment

Rashmi S. Shetty
University of South Florida

An Event Driven Single Game Solution For Resource Allocation In A Multi-Crisis Environment

by

Rashmi S. Shetty

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Nagarajan Ranganathan, Ph.D.
Sudeep Sarkar, Ph.D.
Soontae Kim, Ph.D.

Date of Approval
November 9, 2004

Keywords: crisis management, nash equilibrium, n player game, normal form game, game theory

DEDICATION

To my Parents and my Sister

ACKNOWLEDGEMENTS

I would like to express my gratitude to my major professor, Dr. N. Ranganathan, for his guidance and support through every step of my work. I would also like to thank Narender Hanchate for his valuable insights and advise. Last but not the least, I would like to thank my family and friends who have always been a constant source of support.

TABLE OF CONTENTS

LIST OF TABLES iii

LIST OF FIGURES iv

ABSTRACT vi

CHAPTER 1 INTRODUCTION 1

1.1 Crisis Management 1

1.2 Crisis Management Systems/Agencies 4

1.2.1 FEMA - Federal Emergency Management Agency 4

1.2.2 INFOSPHERE – Sense and Respond Systems 7

1.2.3 CMS (Crisis Management System) 8

1.3 Motivation and Contributions 8

1.4 Thesis Outline 9

CHAPTER 2 RELATED WORK 10

2.1 Types of Algorithmic Approaches to Resource Allocation 12

2.1.1 Dynamic Programming 12

2.1.2 Integer Programming 14

2.1.3 Lagrange Multiplier Method 15

2.1.4 Simulated Annealing 16

2.1.5 Genetic Algorithms 18

2.1.6 Branch and Bound 19

2.1.7 Greedy Algorithm 20

2.1.8 Tabu Search 21

2.2 Why Game Theory? 21

2.3 Game Theoretic Concepts 22

CHAPTER 3 FORMULATION OF A GAME 26

3.1 Crisis Scenario 26

3.2 Modeling of Crisis Scenario as a Noncooperative Strategic Game 28

3.3 Notations 29

CHAPTER 4 OPTIMAL ALLOCATION MODEL USING NASH EQUILIBRIUM 31
4.1 Structure of a strategy 31
4.2 Generation of Strategies 33
4.2.1 Pruning of Strategies 34
4.2.2 Cost Function 36
4.3 Payoff Modeling 40
4.3.1 Creation of Payoff Matrices 40
4.3.2 The Payoff Function 41
4.4 Algorithm to Approximate Nash Equilibrium 43
4.5 Software Implementation 45
4.5.1 System Input and Output 46
4.5.2 Object-Oriented Design 47
4.5.3 Overview of Classes and Functions 48

CHAPTER 5 EXPERIMENTAL RESULTS 52
5.1 Fairness 53
5.2 Execution Time 57
5.3 Statistical Significance of Experimental Results for Execution Time 60

CHAPTER 6 CONCLUSIONS AND FUTURE WORK 62
6.1 Conclusions 62
6.2 Future Work 63

REFERENCES 64

LIST OF TABLES

Table 3.1	Resource Types and Availability	27
Table 3.2	Crisis Types and Requests	27
Table 3.3	Crisis Priorities	27
Table 3.4	Time (in minutes) Taken to Reach Crises	28
Table 4.1	Overview of Classes Used	48
Table 5.1	Fairness Measures	55
Table 5.2	Regression Analysis Results	60

LIST OF FIGURES

Figure 1.1	Crisis Scenario	2
Figure 1.2	FEMA – SLG 101 The Planning Process	5
Figure 1.3	FEMA – SLG 101 Resource Management Organization	6
Figure 1.4	Infosphere – An Overview	7
Figure 2.1	Criteria for Resource Allocation	11
Figure 2.2	Generic Algorithm for Dynamic Programming	13
Figure 2.3	Structure of Simulated Annealing Algorithm	17
Figure 2.4	Structure of a genetic algorithm	18
Figure 2.5	Flow of Branch and Bound Algorithm	20
Figure 4.1	Formulation of a Strategy	32
Figure 4.2	Generation of Strategies	33
Figure 4.3	Algorithm to Generate Strategies	34
Figure 4.4	Recursive Algorithm to Generate Strategy Set for each Crisis	35
Figure 4.3	Flowchart Illustrating Ordering of Strategies	26
Figure 4.4	Normal Form Game Representation	27
Figure 4.5	Flowchart Illustrating Ordering of Strategies	37
Figure 4.6	Algorithm to Add a Strategy to a Strategy Set	38
Figure 4.7	Normal Form Game Representation	39
Figure 4.8	Structure of Payoff Matrices for Crises	41
Figure 4.9	Algorithm to Approximate Nash Equilibrium	44
Figure 4.10	Overview of Crisis Management System	45

Figure 4.11	Workflow Model of the Proposed System	51
Figure 5.1	Effect on Execution Time Due to Increase in Number of Resource Centers	58
Figure 5.2	Effect of Increasing Number of Crises on Execution Time	58
Figure 5.3	Percentage Increase in Computation time of Nash Equilibrium	59

AN EVENT DRIVEN SINGLE GAME SOLUTION FOR RESOURCE ALLOCATON IN A MULTI-CRISIS ENVIRONMENT

Rashmi S. Shetty

ABSTRACT

The problem of resource allocation and management in the context of multiple crises occurring in an urban environment is challenging. In this thesis, the problem is formulated using game theory and a solution is developed based on the Nash equilibrium to optimize the allocation of resources to the different crisis events in a fair manner considering several constraints such as the availability of resources, the criticality of the events, the amount of resources requested etc. The proposed approach is targeted at managing small to medium level crisis events occurring simultaneously within a specific pre-defined perimeter with the resource allocation centers being located within the same fixed region. The objective is to maximize the utilization of the emergency response units while minimizing the response times. In the proposed model, players represent the crisis events and the strategies correspond to possible allocations. The choice of strategies by each player impacts the decisions of the other players. The Nash equilibrium condition will correspond to the set of strategies chosen by all the players such that the resource allocation optimal for a given player also corresponds to the optimal allocations of the other players. The implementation of the Nash equilibrium condition is based on the Hansen's combinatorial theorem based approximation algorithm. The proposed solution has been

implemented using C++ and experimental results are presented for various test cases. Further, metrics are developed for establishing the quality and fairness of the obtained results.

CHAPTER 1

INTRODUCTION

Over the decades, crisis management has developed into a complex and multifaceted issue. The nature of a crisis or a disaster ranges from natural disasters like hurricanes and earthquakes to man-made crises like plane crashes, terrorist attacks, willful acts of mass destruction, industrial accidents, etc. With the development in infrastructure, the impact on a community in terms of damage to property and loss of lives due to the occurrence of a crisis necessitates the need for an organized and effective crisis management system. The scope of a crisis management system includes but is not limited to risk analysis, sensing, responding, monitoring and mitigating the effects of a crisis.

1.1 Crisis Management

Every community is equipped with an array of emergency response units to cater to varied crisis scenarios. Effective recovery from a crisis requires immediate deployment of requested units to the crisis locations. The complexity of this arises from the heterogeneity of emergency response units, e.g., fire engines, ambulances and police cars. Furthermore, these units are distributed over a wide area and control-led by multiple organizations. Each crisis is unique in its severity, request for number and type of resources, location and potential growth. In the event of multiple simultaneous crises, it is critical to ascertain the severity of each crisis and allocate the optimum number and appropriate type of emergency units to each location. Depending on the

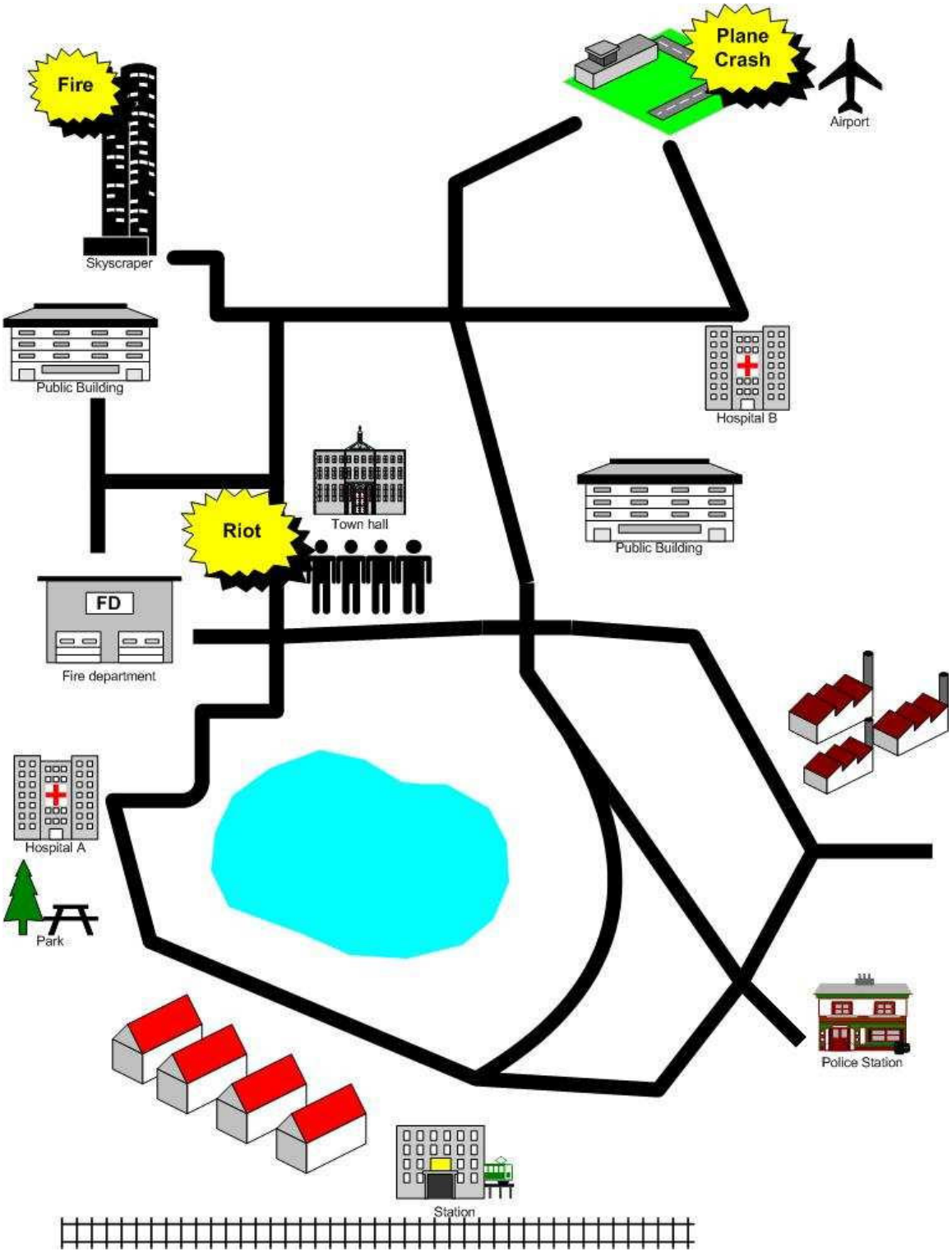


Fig 1.1 Crisis Scenario

location and nature of the crisis, there may be additional requests for emergency units to prevent further deterioration of a situation. Although, some crises may be more critical than others, all of them need to be serviced immediately to prevent spawning of additional crises and further damage. The main aim of allocation of emergency services is maximization of the utility of existing and available emergency response units and minimization of response time to mitigate the effects of one or many crises. Fig 1.1 illustrates a scenario to facilitate further understanding of the situation.

Let us assume a hypothetical crisis scenario in a city.

- A plane crashes as it lands in the airport. It is a small passenger plane with 30-35 people including the crew. The crash has affected 2-3 planes parked at their terminals at the airport.
- A fire breaks out in an apartment on the sixth floor of a high-rise building trapping the people within the apartment. The fire is slowly spreading to other apartments on the same floor and the one above.
- A demonstration in front of the town hall has turned into a riot. A few members of the riot have become violent and are throwing inflammable objects all around the area.

Each of the above incidents qualifies as a crisis although each varies in its degree of criticality and the number of resources it requires. The plane crash has the highest priority because of the casualties and the possibility of worsening. The airplanes have fuel in them which is highly inflammable and as other planes have been affected, the explosion has to be controlled before it spreads to the airport and causes further damage. The fire is next in priority and has to be controlled before it consumes the entire building and causes further damage to life and property. The riot comes next in terms of criticality. In the scenario, we consider three types of resources available – ambulances from the two hospitals, police cars from the police station and fire engines from the fire department. The plane crash and the fire scene would require fire engines and

ambulances. Both the locations might require a few police cars to regulate the population and direct people to safety. The riot scene would require a lot of police cars to control the crowd, maintain law and order and prevent stampedes. It would also need a few ambulances for riot casualties and fire engines to control fires started by rioters. A lot of factors like availability, distance, traffic conditions, etc determine how many resources are dispatched to the crisis locations and from which resource center. For example, there are two hospitals, Hospital A and Hospital B. It is important to note that Hospital A is closer to the riot than Hospital B. Hence, it is more practical to send more ambulances from Hospital A to the riot than Hospital B. However, the plane crash would receive more ambulances in total than the riot due to its higher criticality. It is highly possible that there may not be sufficient number of ambulances and fire engines as compared to the requests made by the crisis locations. Hence it is critical to make an optimal allocation of resources as under or over utilization of resources could cost lives. The allocation has to weigh in factors such as criticality of crises, request and availability of resources, distance and number of resource centers.

1.2 Crisis Management Systems/Agencies

Several agencies have been set up and systems have been designed to monitor and mitigate the effects of crises. We will review some of the salient features of these agencies and systems.

1.2.1 FEMA - Federal Emergency Management Agency

The Federal Emergency Management Agency - a former independent agency that became part of the new Department of Homeland Security in March 2003 – has the task of responding to, planning for, recovering from and mitigating against disasters. FEMA can trace its beginnings to the Congressional Act of 1803, a piece of disaster management legislation, which provided assistance to a New Hampshire town following an extensive fire. In the century that followed, ad hoc legislation was passed more than 100 times in response to hurricanes, earthquakes, floods and

other natural disasters. Over the course of the last few years, nuclear power plants, transportation of hazardous substances and civil defense responsibilities were added to this agency further compounding the complexity of emergency management.

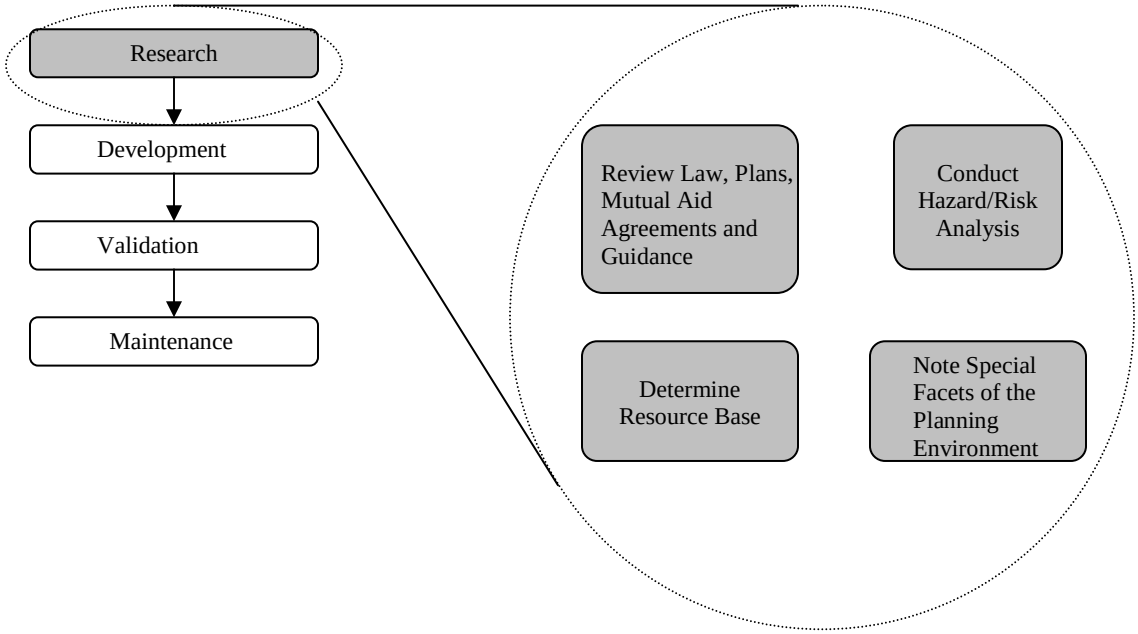


Fig 1.2 FEMA – SLG 101 Planning Process (Chapter 2)

According to Federal Emergency Management Agency (FEMA) Strategic Plan Fiscal Years 2003 – 2008:

- Crisis/Disaster: Broadly defined to include disasters and emergencies that may be caused by any natural or man-made event
- Response: Conducting emergency operations to save lives and property, including positioning emergency equipment and supplies; evacuating potential victims; providing food, water, shelter, and medical care to those in need; and restoring critical public services

- Recovery: Rebuilding communities so individuals, businesses, and governmental infrastructure can function on their own, return to normalcy, and are protected against future hazards

Fig 1.2 shows an excerpt from the State and Local Guide (SLG) 101: Guide for All-Hazard Emergency Operations Planning – Chapter 2. These are guidelines to emergency preparedness. The grey boxes indicate those tasks which are relevant to resource allocation. Any proposed solution would require complete information about the resource base, topography, jurisdiction

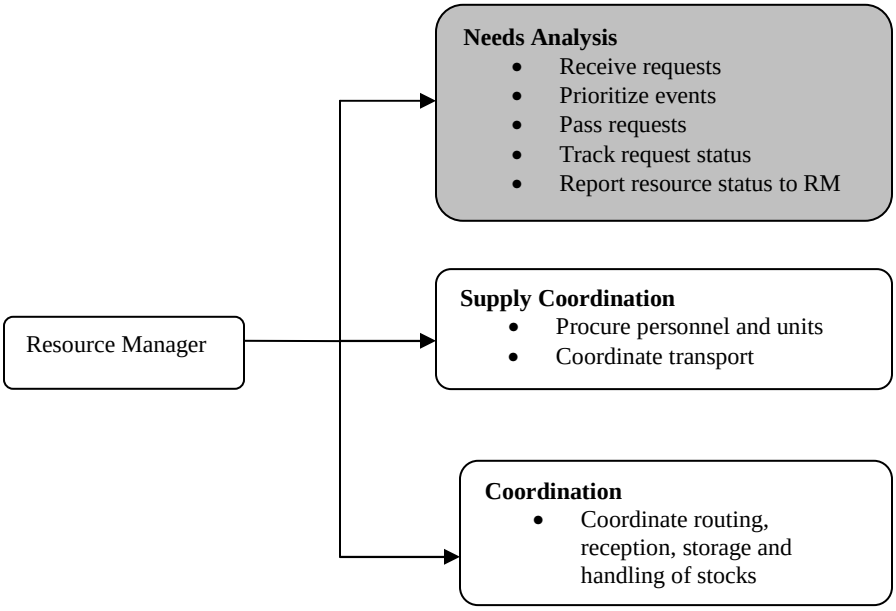


Fig 1.3 FEMA – SLG 101 Resource Management Organization (Chapter 6)

and classification of priority. After collecting information about resources and possible hazards, a system needs to have guidelines on distribution of resources. Fig 1.3 gives a brief outline of the resource management organization as proposed by Chapter 6 of FEMA – SLG 101. The shaded region has relevance to our work in terms of acquiring real-time resource updates, prioritization of events and dispatching resources.

1.2.2 INFOSPHERE – Sense and Respond Systems

The Caltech Infospheres Project [72] is devoted to research on compositional systems or, systems built from interacting components. One of the applications of this project includes a "sense and response" system. The basic purpose of the system is to hold a repository of data from multiple sources and normalize it to a standardized vocabulary. Certain conditions are specified that generate system alerts. The system monitors data from various institutions and when the specified condition is met, alerts are sent securely to the destination. The system can be programmed for specific applications, for e.g., an airplane switches to a different mode when the system detects an equipment malfunction. The data sources monitored for specific applications are immense in terms of volume, speed, heterogeneity and distributed nature. The proposed system is evaluated on the basis of – frequency of errors, response time, computational resources consumed, scalability and ease of adaptation. Fig 1.4 gives an overview of the control flow of the proposed system.

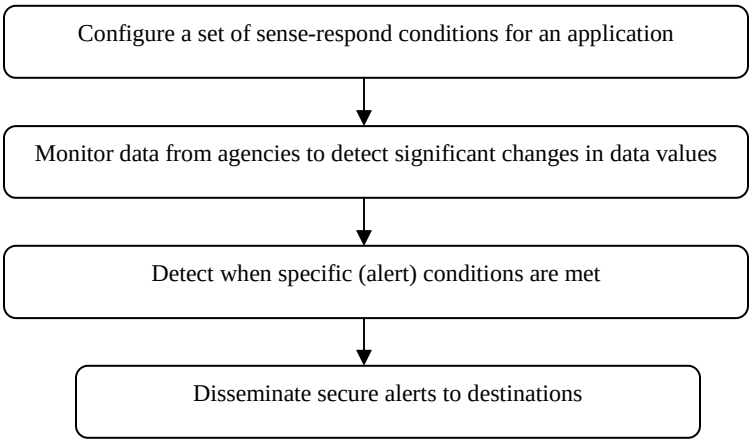


Fig 1.4 Infosphere – An Overview

1.2.3 CMS (Crisis Management System)

CMS [71] is a powerful crisis management software developed by Applied Science Associates, Inc. It has been designed as a tool for marine emergency response managers to model the impacts and biological effects of spill or a disaster. It can be used for training and simulation exercises, cost-benefit analysis and facilitate real-time response to marine disasters. Its interface has been designed for an oil spill, chemical spill, search and rescue mission, marine emergency and nuclear disaster. All the machines part of CMS are connected to a single resource database and enable immediate access to resource information. It is equipped with a Geographical Information System (GIS) to obtain real-time environmental information. It is used for management and distribution of information for an organization handling response and recovery.

1.3 Motivation and Contributions

Multi-crisis management is a complex problem encompassing sensing, responding and recovering from crises. It is important to set good precedents for future occurrences to enable a better level of preparedness and response to crises to minimize destruction of life and property. In the previous section, we observe systems and agencies designed to improve and enhance the ability of emergency response managers to make effective crisis management decisions. They are primarily protocols or guidelines (like FEMA) for risk analysis, resource base analysis, and coordination of personnel and resource units. Some others are alerting mechanisms (like INFOSPHERE) used to monitor the activities of systems (a house, an airplane, city, etc). Systems like CMS are tools to monitor crises and provide real-time updates on them and the resources sent to mitigate their effects. While ample work has been done in collecting data regarding resources and classification of crises, there has not been significant work to automate the allocation of resources to crisis situations. The actual allocation of resources is made manually based on predefined guidelines and protocols and real-time information gathered from the scenes and is susceptible to human error.

This work proposes a model that uses the same information and enumerates feasible allocation strategies and determines an optimal set of strategies for managing each crisis. The proposed application is based on an optimization algorithm to obtain the best possible allocation that benefits all crisis locations. The multi-crisis scenario is modeled as a strategic game [22, 23] where the crisis locations are considered as players and each possesses a set of strategies that correspond to allocations from different resource centers. All the crises compete for the same set of resources and their adversarial nature is used to model them as players in a noncooperative game where each player tries to maximize his own utility. Each strategy of a crisis has an associated cost which is a function of resources contributed and the time taken to service a crisis' request. The strategy sets are inputs to the optimization algorithm. The algorithm uses an objective function that associates a payoff with each allocation based on priority of a crisis, its request, number of resources available and time taken to reach the crisis. The optimization algorithm produces as an output an allocation of resources from each resource center to each of the crises. It is based on the principle of the Nash Equilibrium [22, 23, 69, 70] which provides a socially viable solution whereby any player which deviates from the equilibrium strategy will earn less than if it remained in its current strategy.

1.4 Thesis Outline

The thesis is organized as follows: Chapter 2 briefly looks at other algorithmic approaches to solving resource allocation problems and the approach used in this work. We explain the reasons for adopting game theoretic concepts to our problem. Chapter 3 introduces us to the formulation of the problem and the algorithmic design. We explain the constraints of the problem and the objective function used. Chapter 4 presents the experimental results followed by conclusions and an overview of future work in Chapter 5.

CHAPTER 2

RELATED WORK

Technological advances over the last few decades have drastically decreased the time delay between theoretical progress and its practical impact. One such subject which has been widely researched is resource allocation and it has found numerous applications in load distribution [10], production planning [11], computer scheduling [12] and many other areas. Since Koopman's work [5] on the optimal distribution of effort in 1953, a significant number of papers [2, 3, 4, 6, 7] have been published on the subject. The allocation of resources is an optimization problem with the constraint – given a fixed quantity of a resource type, determine its allocation to a set of activities, such that the objective function or (in our case) the payoff function is optimized. Formally, this can be stated as follows [1]:

Resource: minimize $f(x_1, x_2, ..., x_n)$

Subject to $\sum_{j=1}^{n} x_j = N$,

where $x_j \geq 0$, $j = 1, 2, \ldots, n$

Here, x_j represents the amount of the resource that is allocated to activity *j* and *f* represents the objective function. *N* represents the total amount of the resource available and *n*, the total number of activities. The objective value $f(x_1, x_2, ..., x_n)$ could be a cost, a reward, profit or loss as a result of the allocation. If the resource is divisible, it can be represented by any nonnegative value

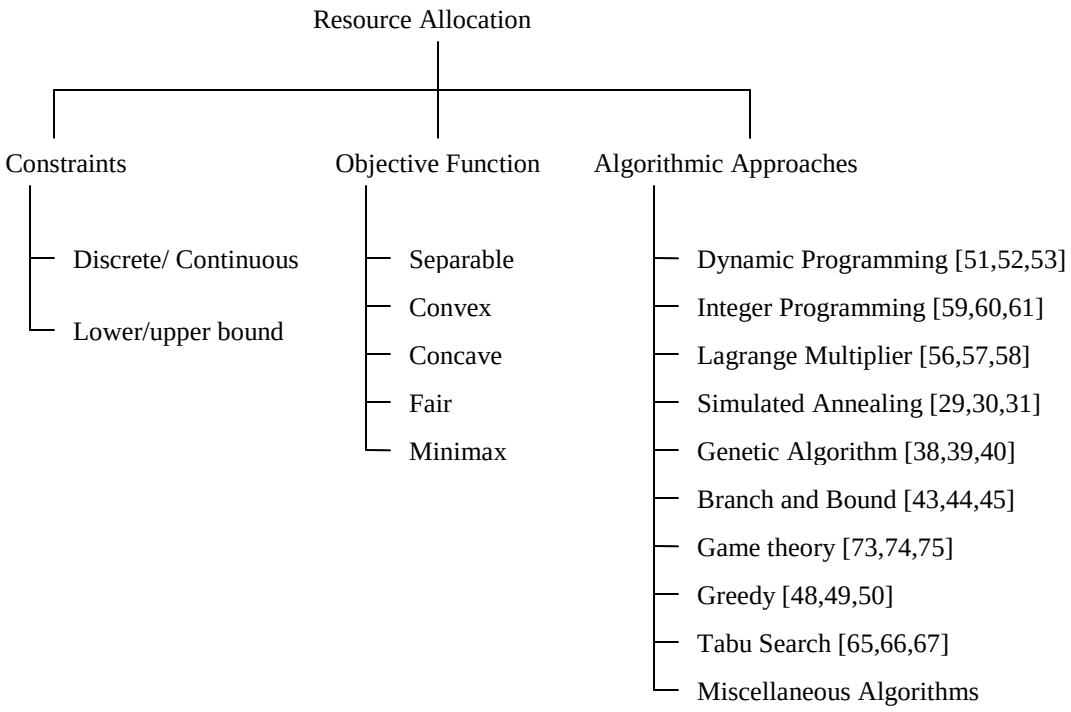


Fig 2.1 Criteria for Resource Allocation

and x_j is a continuous variable. The resource may not be divisible if it represents persons,

$$l_j \le x_j \le u_j, \qquad j = 1, 2, \ldots, n$$

vehicles, parts and so on. In the context of this work, x_j is a discrete variable that can be represented only in nonnegative integer values. Sometimes, lower bounds (other than/greater than 0) and/or upper bounds are imposed such that, it is required to allocate atleast l_j and at most u_j resources to activity *j* [1].

The choices of algorithms for resource allocation depend on how efficiently an objective function can be exploited. Some of the typical forms of objective functions are [1]:

- Separable: $\sum_{j=1}^{n} f_j(x_j)$, where each f_j is a function of one variable
- Convex [9]: A function *f(x)* is convex if, on an interval [x, y], for any points a_1 and a_2 in the interval [x, y], $f[1/2(x_1 + x_2)] \leq 1/2[f(x_1) + f(x_2)]$.
- Concave [9]: A function *f(x)* is concave if, on an interval [x, y], for any points a_1 and a_2 in [x, y], the function *–f(x)* is convex on that interval.
- Minimax: Minimize $\max_j f_j(x_j)$; and Maximin: maximize $\min_j f_j(x_j)$
- Fair: Minimize $g(\max_j f_j(x_j), \min_j f_j(x_j))$, where $g(u, v)$ is nondecreasing (respectively, nonincreasing) with respect to u(respectively, v)

2.1 Types of Algorithmic Approaches to Resource Allocation

Several algorithmic solutions and their generalizations have been proposed to obtain optimal solutions to the resource allocation problem. In this chapter, we will examine some of the widely used algorithms to solve the resource allocation problem and examine how game theory is well suited for modeling the solution in our case. Although we examine algorithmic approaches to solving discrete and continuous resource allocation problems, we emphasize discrete algorithms due to its relevance in this work.

2.1.1 Dynamic Programming

Dynamic programming typically applies to optimization problems where we make a set of choices to obtain an optimal solution. There may be several solutions to obtain the optimal value. A dynamic-programming algorithm can be categorized into four steps:

- Define the structure of an optimal solution
- Recursively define an optimal solution
- Compute an optimal solution in a bottom-up manner

- Build an optimal solution from the information obtained in the previous steps

In [13], a dynamic programming formulation is used for a time-optimal multi-agent task assignment problem. Here, *m* tasks are to be assigned to *n* agents, with $m \leq n$, and one agent can perform only one task. A task assignment algorithm for a global optimal task assignment is obtained based on a problem specific recurrence relation derived using the Principal of Optimality [14]. Next, a dynamic programming styled time-optimal task assignment algorithm is constructed since each stage of the algorithm is based on the recurrence relation derived earlier. Dynamic programming is similar to the divide-and-conquer problem. However, the latter approach is not suitable for cases when there are common subproblems as it solves them repeatedly. Dynamic programming solves each subproblem just once and saves it in a table and thereby avoids recomputation every time the subproblem is encountered. However, the disadvantage of dynamic programming is that when it is applied to any multistage optimization problem, the dimensionality explodes when there are several state variables and each of them has large discretization

- Define a non-empty state space X with a finite set of states
- Define a finite set of actions, U(*x*) that can be applied from a state $x \in X$
- Let $k \in \{1 \ldots K+1\}$ where *k* is a stage. We assume that K is larger than the longest optimal path between any two states $\in X$
- Let F = K + 1 where F is the final stage
- $x_{k+1} = f(x_k, u_k)$ for $x_k \in X$ and $u_k \in U(x_k)$ where *f* denotes the state transition equation
- Let x_1 denote the initial state and x_g denote the state we want to reach or the goal state
- Define an additive loss function L,

$$L = \sum_{k=1}^{K} l(x_k, u_k) + l_F(x_F)$$

where $l_F(x_F) = 0$, if $x_f = x_G$ and $l_F(x_F) = \infty$ otherwise

- Define a termination action $u_T \in U(x)$, such that if u_T is applied to x_k, then the action is repeatedly applied until stage *K*, and the state remains in x_k until the final stage. Also, $l(x_k, u_T) = 0$ for any *k* and x_k
- Find $u_1 \ldots\ldots u_K$ that minimizes *L*

Fig 2.2 Generic Algorithm for Dynamic Programming [15]

levels. In our case, the state variables would correspond to the number of units in each resource center. Also, the computational cost of the proposed algorithm grows rapidly with the number of agents making it infeasible for a large number of agents. Inspite of the curse of dimensionality, we find dynamic programming being applied to a variety of resource allocation scenarios as described in [51, 52, 53, 54, 55].

2.1.2 Integer Programming

Many optimization problems can be expressed as linear or nonlinear programming problems. A linear program is a problem which can be expressed as follows:

Minimize cx

Subject to $Ax = b$ $x \geq 0$

where x is a vector of variables to be solved and A is the matrix of coefficients, and c and b are vectors of known coefficients. "cx" is referred to as the objective function, and the expression "$Ax = b$" is called a constraint. A nonlinear program is a problem of the form,

Minimize $f(x)$

Subject to $gi(x) = 0$ for $i = 1, \ldots, m1$ where $m1 \geq 0$

$hj(x) \geq 0$ for $j = m1 + 1, \ldots, m$

$f(x)$ is a an objective function consisting of several variables and the other two functions are constraints. If the unknown variables are required to be integers, as is the case with this work, then the problem is referred to as integer programming. If the problem requires only some of the variables to take on integer values, it is called mixed integer programming. Although this is more realistic, it is harder to solve. Integer programming techniques can be applied over a substantial range of problem sizes and applications. The work in [16] integer linear programming is used to improve bandwidth efficiency in networks using a segment protection algorithm. An active path (AP) in a network is divided into several active segments (AS) which are protected by backup segments (BS). In case of a failure, the traffic is rerouted through a BS. Integer linear

programming is used to determine an optimal partition of a given AP into ASs and find the corresponding BSs. The work in [17] uses integer linear programming to obtain an optimal allocation of registers for general purpose processors and embedded systems. Although integer programming techniques are known to provide optimal solutions, in both the works above it has been found that they can be used for medium sized and not large networks (in the first case) and solution times are slow (in the second case). Integer programs are undecidable in the worst case and in some cases found to be NP-Hard. [59, 60, 61, 62, 63, 64] are some works which use pure integer and mixed integer programming techniques for their resource allocation problems.

2.1.3 Lagrange Multiplier Method

In mathematical optimization problems, Lagrange multipliers are used to deal with problems with constraints. They are used to find the maxima or minima of a multivariate function subject to a constraint [1].

Optimize $f(x, y)$

Subject to $g(x, y) = 0$

The Lagrangian is written as, $L = f(x, y) + \lambda\, g(x, y)$ where λ is a constant called the Lagrange multiplier. According to Lagrange's multiplier method, the simultaneous conditions are,

$$\left(\frac{\partial L}{\partial x}\right)_y = 0; \qquad \left(\frac{\partial L}{\partial y}\right)_x = 0;$$

The goal is to find the maximum and minimum values taken on by f along the curve with the constraint on the points, $g(x, y) = 0$. The Lagrangian approach treats all variables and constraints in a symmetrical fashion so that problems involving numerous variables and constraints can be neatly organized. [18] uses Lagrangian methodology to schedule and allocate resources in a manufacturing unit. The objective is to efficiently use limited resources to meet dynamic customer requirements. Factories use a flexible manufacturing system by using production layouts to simplify production flow lines and increase productivity. Scheduling is used to decide

when to set up a cell for a production lot and the quantity of machines to allocate to the cell. The machine capacities, processing time and the machine type are quantified as constraints. The model uses Lagrange relaxation and forms a dual function by relaxing complicating constraints with Lagrangian multipliers. The original problem is divided into subproblems which are easier to solve. The model produces schedules which are 16%-29% within optimal. [19] is another work using a similar approach to optimally allocate resources in a distributed computing environment. Lagrangian relaxation and subgradient methods are applied to solve this problem. It was observed that an optimal solution would occur in an earlier iteration without converging. It has been observed that in the works above solutions to subproblems are not feasible and additional heuristics are applied to arrive at feasible schedules. [56, 57, 58] are some other examples of the application of the Lagrange multiplier method to the resource allocation problem.

2.1.4 Simulated Annealing

This algorithm is based on that of Metropolis et al. [9] to find an equilibrium configuration of a collection of atoms at a given temperature. In 1973, Pincus et al. [26] drew an analogy between this algorithm and mathematical minimization. It was proposed as an optimization technique for combinatorial problems by [27]. Simulated annealing is a random search technique which has the advantage of not getting trapped in local minima. It accepts changes that increase and decrease an objective function *f*. An increase in the change is accepted with the probability *p* [28], where $p = e^{\left(\frac{-\delta t}{T}\right)}$. δt is the increase in *f* and *T* is a control parameter referred to as the "system temperature". [41] suggests that the initial temperature T_0 has a significant impact on the performance of the algorithm. The algorithm requires a problem-specific annealing schedule, i.e. an initial temperature and the rules for lowering it as the search proceeds. Figure 2.2 shows the structure of a simulated annealing algorithm. One of the major problems in the implementation of simulated annealing lies in the difficulty of drawing an analogy between *T* and a free parameter in

a real-life problem. Furthermore, staying out of local minima is dependent on the choice of an annealing schedule, number of iterations performed for each temperature and the decrements of temperature towards the cooling process. [29, 30, 31, 32, 33, 34] are some recent examples of the practical applications of simulated annealing as an optimization algorithm.

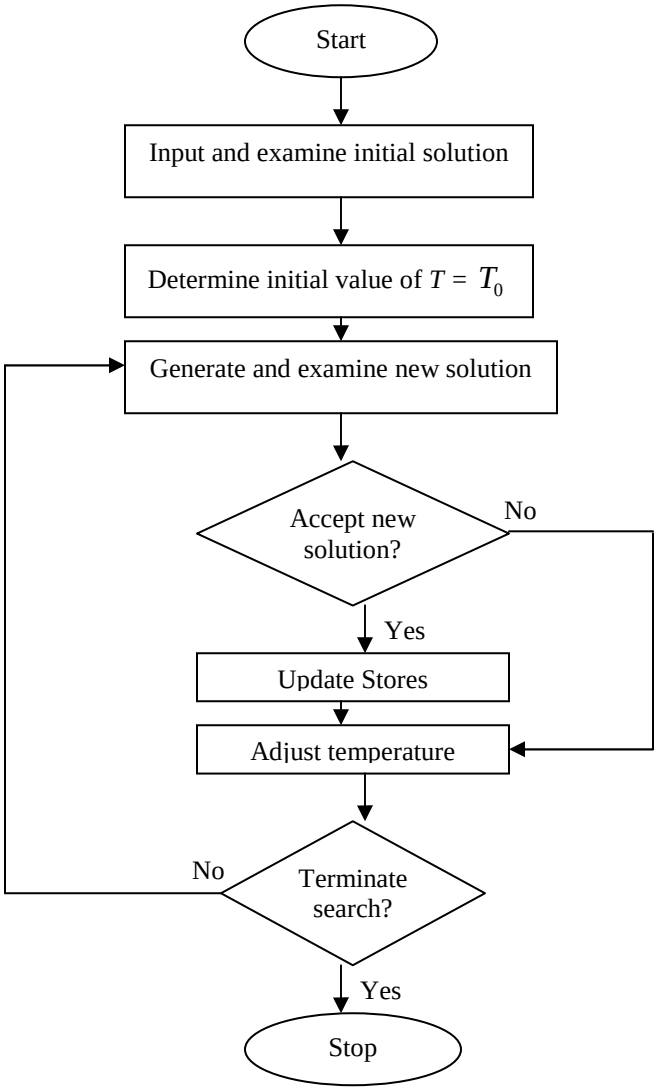


Fig 2.3 Structure of Simulated Annealing Algorithm [28]

2.1.5 Genetic Algorithms

The framework for genetic algorithms lies in the natural evolution of species searching for beneficial adaptations in a changing environment. Although the information encoded in the chromosomes of individual members changes by random mutation, it is essentially a combination of chromosomal material during breeding. In 1975, the incorporation of the principles of evolution into optimization routines was formally established in [36]. To use a genetic algorithm, we need to represent the solution as a genome (or chromosome). The algorithm takes as an input an initial population of solutions and applies the genetic operators (mutation, crossover) to evolve and find an optimum solution. The main aspects of applying genetic algorithms to real-life

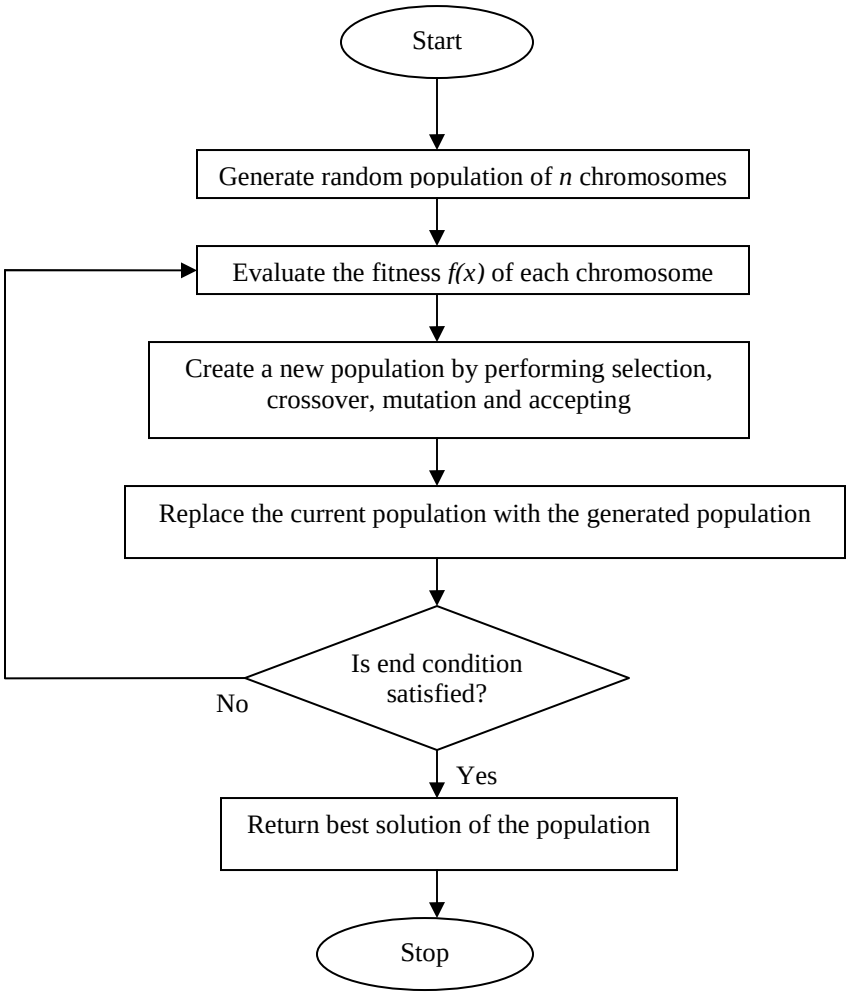


Fig 2.4 Structure of a genetic algorithm

problems are: (i) defining an objective function; (ii) obtaining a genetic representation of the solution; (iii) defining the genetic operators. The efficiency of a genetic algorithm is dependent on the control parameters: (i) initial population, (ii) size of the population, *N*, iii) crossover probability, *Pc*, (iv) mutation probability, *Pm*. Genetic algorithms are similar to simulated annealing as both use probabilistic transition rules and use objective function information and not derivatives [28]. Genetic algorithms, although computationally expensive, can be easily parallelized as the evaluation of an objective function and constraints can be done simultaneously for a whole population. The figure 2.3 shows the structure of a genetic algorithm. [38, 39, 40, 41, 42] are some of the recent works which use genetic algorithms to obtain optimal solutions for the resource allocation problem.

2.1.6 Branch and Bound

Branch and bound is another algorithm used to solve optimization problems. It searches the entire solution space for the best solution. However, as the number of possible solutions increases exponentially, it becomes infeasible to enumerate all of them and hence we use bounds for the function to be optimized. The algorithm consists of three main parts [37]:

- A bounding function to determine the lower bound for the best solution in the given subspace of the solution
- A strategy for determining the next solution subspace to be analyzed. A branching rule to be applied if a subspace after analysis cannot be excluded and further subdivide the subspace into two or more subspaces
- The performance of the branch and bound algorithm depends to a great degree on the initial search space fed to the algorithm.

Convergence is ensured if the size of each generated subspace is smaller than the original one. [43, 44, 45, 46] are works which employ the branch and bound algorithm to find optimal solutions to their versions of the resource allocation problem. Although branch and bound

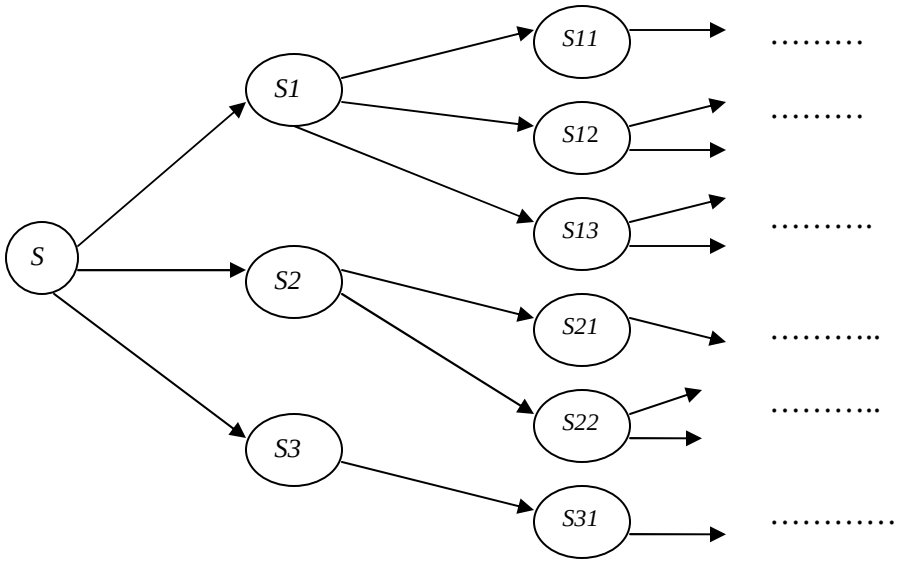


Fig 2.5 Flow of Branch and Bound Algorithm

algorithms are favorable for discrete and continuous global optimization problems, they have high memory requirements.

2.1.7 Greedy Algorithm

A greedy algorithm is one which follows a problem solving heuristic of making a locally optimum choice in the hope of obtain a globally optimum solution. Greedy algorithms do not always yield optimal solutions as they do not exhaustively examine all the possible solutions of a search space. The basic elements of a greedy algorithm are [47]:

- A solution space from which the solution is created
- A selection function to choose the next best element to be added to the solution
- A feasibility function to examine an element's eligibility to be added to the solution
- An objective function to calculate the value of the (full or partial) solution obtained
- A solution function to determine if the complete solution has been reached

Greedy algorithms are rarely used to obtain optimal solutions and usually form the basis of a heuristic approach. Even though there maybe problems which can be solved using the greedy

approach, establishing their optimality is non-trivial. [48, 49, 50] are examples of works use the greedy approach to solve resource allocation problems. Typically, greedy approaches are not used because of their unreliability in providing optimal solutions as is proved in the case of the work in [50].

2.1.8 Tabu Search

A Tabu search is a global optimization algorithm which is a meta-heuristic imposed on another heuristic. In 1977, Fred Glover introduced this approach of moving through a solution space and using memory techniques to avoid cycling. The algorithm records moves in a Tabu list and penalizes it if it takes the solution to a point in the solution space that has been previously visited. Hence, the algorithm avoids getting trapped in cycles. Tabu search is still an evolving and highly researched technique of optimization. Although Tabu search provides comparable or superior solutions to optimization problems, it does not guarantee optimality. Also, the construction of a Tabu list to keep a record of the moves is heuristic. [65, 66, 67, 68] are some of the recent applications of Tabu search to practical problems.

2.2 Why Game Theory?

Game theory is a tool for modeling and analyzing conflict and cooperation between decision makers called players [20]. Such a situation occurs when multiple decision makers with different objectives act on a system or share resources. Game theory is considered to have been formalized with the publishing of von Neumann and Morgenstern's *The Theory of Games and Economic Behaviour* in 1944. Game theory provides a natural framework for the modeling of the crisis scenario in this thesis. In the context of this work, we use noncooperative games due to the competitive nature of the players (or, crises in our case). Two or more crises compete for a limited number of emergency units from various centers. They have a finite set of actions or allocation strategies available to them, the choice of which leads to a well defined numerical

payoff associated with the each combination of strategies and for each crisis. [21]The strategy selected by a crisis depends on three parameters:

- Strategy space which consists of the set of strategies (allocations) available to the crisis
- Information about the other crises (priority, distance from resource center)
- Payoff or utility function which quantifies the satisfaction a user can get from an outcome

Every player attempts to maximize his gain in the game. A significant aspect of game theory is that each player's decision is based on the decision of every other player and hence, each player can optimize his gain with respect to every other player. This is quite useful in modeling the crisis scenario where the overall optimization is feasible only *if each crisis has been satisfied with respect to all other crises*. The adversarial nature of the game and the interdependence of each player's objective function on the decisions of the other players require the resource allocation strategy to take into account all the other players' objective functions. The Nash solution in game theory provides an equilibrium solution taking into account the objective functions of all players. A significant aspect of game theory is that it has been proven that a finite noncooperative game has at least one Nash equilibrium [22]. This is motivation for us to use the Nash equilibrium as we are guaranteed an equilibrium strategy set for a crisis scenario.

2.3 Game Theoretic Concepts

Games, as represented in game theory, consist of four essential elements – players, actions, payoffs, and information. These constitute the rules of the game. Depending on the *information* available, *players* try to maximize their *payoffs* by choosing *strategies.*

- Players are the individuals/ entities who make decisions. Each player's goal is to maximize his utility by a choice of actions
- An action or strategy of player *i*, denoted s_i, is a decision he can make. Player *i*'s action $S_i = \{ s_i \}$, is the entire set of actions available to a player.

- By player *i*'s payoff $\pi_i(s_1,...,s_n)$, we mean the utility received by player *i* after all players have decided their strategies and the game has been played; or

  The expected utility received as a function of the strategies chosen by the player *i* and the other players

There are two kinds of strategies: pure and mixed [22]

A pure strategy maps each of a player's possible information sets to one action.

$$s_i : \omega_i \rightarrow a_i$$

A mixed strategy maps each of a player's possible information sets to a probability distribution over actions

$$s_i : \omega_i \rightarrow m(a_i) \text{ Where } m \geq 0 \text{ and } \int_{A_i} m(a_i) da_i = 1$$

Here ω_i refers to the information set.

A strategy combination $s = (s_1,...,s_n)$ is an ordered set consisting of one strategy from each of the *n* players. Every player in a game maximizes its payoff and arrives at an equilibrium state. An equilibrium $s^* = (s_1^*,...,s_n^*)$ is a strategy combination of the best strategy for each player in an n-player game. The strategy combination s^* (a set of strategies) is a Nash equilibrium is, if any player which deviates from its strategy will earn less than if it remained in its current strategy. Formally, this can be stated as,

$$\forall i, \pi_i(s_i^*, s_{-i}^*) \geq \pi(s'_i, s_{-i}^*), \forall s'_i$$

The inputs required to formulate Nash equilibrium are:

- Strategies available for each player
- Number of players in a game

Some of the major categories of classifying games are as follows [25]:

- Number *n* of players – A game can consist of one, two or *n* players
- Number of strategies of players – This may be finite, a discrete infinite or a continuum
- Zero-sum, constant-sum or general-sum – In a zero-sum game, the numerical payoffs to the player after any possible play of the game sum up to zero. A constant-sum game is one where the sum of all payoffs to the player is the same for any outcome. A general-sum game includes the other two.
- Cooperative and noncooperative - In cooperative games, players communicate with one another and can make binding commitments as opposed to noncooperative games
- Complete and incomplete information
- Single-stage or multi-stage games

Non cooperative games can be further classified into strategic (normal form) games and extensive form games.

- A normal form game is a game of complete information played between *n* players, each having a strategy set, S_i and a payoff function p_i where $p_i : \prod_{i \le n} S_i \to \Re$. If there are a finite number of strategies, we can define a normal form game as a matrix. In such a game, each player simultaneously selects a move $s_i \in S_i$ and receives a payoff $u_i(s_1,...,s_n)$
- An extensive form game is one which can be represented as a connected tree with no cycles and a distinguished node where each node represents a decision made by a player. A function specifies which player moves at a node, what actions are available, and which node comes next for each action [24].

For the modeling of the problem stated in this work, our definition of a finite game is a noncooperative *n*-player game, with each player associated with a finite set of pure strategies; and

corresponding to each player, *i*, a payoff function p_i, which maps all the *n*-tuples of pure strategies into real numbers. A tuple is a set of *n* strategies with each strategy associated with a different player.

CHAPTER 3

FORMULATION OF A GAME

The automation of the allocation of emergency response units is a logical step in crisis management to minimize if not completely eliminate human errors in decision making. In this work, we propose a game theoretic framework to address the problem of allocation of resources to multiple crises. In drawing upon the concepts of game theory and consequently Nash Equilibrium we obtain a framework in which we can address the issue of minimization of response time, maximization of utility and fairness of the allocation. The idea of using a Nash solution in the context of resource allocation is not new and has been implemented in several areas [73, 74, 75, 76, 77, 78]. In this chapter, we illustrate the transformation of a multi-crisis environment into a noncooperative strategic game.

3.1 Crisis Scenario

We revisit the example presented in Chapter 1, Figure 1.1. Consider a hypothetical crisis scenario as described in the figure – a plane crash, a fire and a riot. Tables 3.1 to 3.4 contain information about resource types and availability, crisis requests, crisis priorities and time taken to reach crises from each of the resource centers. In the given scenario, we note that the requirement of crises exceeds the capacity of the resource centers. Typically, if the number of resources available for dispatch were less than or equal to the requests made by crises, the resource manager is left with the task of determining the distribution of resources from resource

Table 3.1 Resource Types and Availability

Resource Centers ➲ / Resource Types ⮋	Hospital A	Hospital B	Police Station	Fire Department	Total Availability
Ambulances	8	10	-	-	18
Police Cars	-	-	16	-	16
Fire Engines	-	-	-	14	14

Table 3.2 Crisis Types and Requests

Resource Types ➲ / Crisis ⮋	Ambulances	Police Cars	Fire Engines
Plane Crash	11	6	10
Fire	8	3	6
Riot	3	9	1
Total Request	22	18	17

Table 3.3 Crisis Priorities

Crisis ⮋	Priority (1 - 10) ⮋
Plane Crash	9
Fire	7
Riot	3

Table 3.4 Time (in minutes) Taken to Reach Crises

Resource Centers ➲ / Crisis ⏻	Hospital A	Hospital B	Police Station	Fire Department
Plane Crash	30	7	31	24
Fire	18	15	33	8
Riot	8	11	13	3

centers between crises based on time taken to reach each crisis. Shortage of resources is not an issue and every crisis is guaranteed satisfaction of its request. However, in the current scenario, the resource manager has the additional task of determining the optimal allocation in the face of shortage and possible starvation of lesser priority crises. For example, there are 10 + 8 = 18 ambulances. However, the total request for ambulances is 11 + 8 + 3 = 22 > 18 ambulances. There are two hospitals, A and B and it is important to note that A is closer to the plane crash B while allocating resources. Not only does a resource manager have to decide how many ambulances have to be dispatched from Hospital A and B, he has to decide on an optimal distribution based on severity and distance from crisis due to the shortage of ambulances. In our methodology, we obtain an optimal solution for each *type* of resource separately, i.e., we determine an allocation for ambulances, police cars and fire engines separately.

3.2 Modeling of Crisis Scenario as a Noncooperative Strategic Game

We apply game theoretic concepts to the crisis scenario resource allocation problem. As mentioned in Section 2.3 of Chapter 2, the main elements of every game are players, actions, payoffs and information.

In our model, each crisis is modeled as a player. The objective of each player is to maximize its utility by choosing actions most beneficial to it The actions, in this case, will be the

allocations to resource centers. For example, the plane crash, fire and riot have requested for 6, 3 and 9 police cars respectively. The action of sending 6 police cars to the plane crash, 2 police cars to the fire and 8 police cars to the riot constitutes an allocation. In the context of this work, we refer to the actions as strategies. The utilities are the payoffs received as a function of actions chosen by other players. The framework rests on the modeling of the payoff function as it captures the effect on other crises when one crisis chooses a particular strategy. The information available to the players corresponds to requests, availabilities, priority and response time.

The conflicting objectives of the crises (players) contribute to the *noncooperative* nature of the game. None of the crises make prior commitments to share or lend resources before the game is played. This is a game of *complete information* as we play the game on the assumption that we have the latest information updates on the resource availability and crisis requests. Also, the information regarding time taken to reach a crisis location is assumed to be accurate. All the crises will make their selections simultaneously in a game and the game can be represented in a matrix where each cell represents a payoff value. Hence the game is a *strategic* or normal form game where all crises have *finite* number of strategies in their sets and the players' actions are mapped to a probability distribution.

3.3 Notations

The following notations are used in this work:

n	Number of crises, $n \in \mathbb{N}$
C	Set of crises, $C = \{C_1, C_2, C_3, ...C_n\}$
m	Number of resource centers, $m \in \mathbb{N}$
R	Set of resource centers, $R = \{R_1, R_2, R_3.....R_m\}$
Q	Set of requirements of all crises, $Q = \{q_1, q_2, ...q_i, ...q_n\}$
q_i	Number of resources requested by crisis C_i where $q_i \in \mathbb{N}$
O	Set of resources available at resource centers, $O = \{o_1, o_2, ...o_i, ...o_m\}$

o_i Number of resources available at resource center R_i where $o_i \in N$

S Set of strategies of all crises, $S = \{S_1, S_2, ...S_n\}$

S_i Set of strategies of crisis C_i, $S_i = \{s_{i,1}, s_{i,2}, ..., s_{i,gi}\}$

g_i Total number of strategies in S_i

$s_{i,j}$ j^{th} strategy of the i^{th} crisis, $s_{i,j} \in S_i$

r_k Number of resources contributed by k^{th} resource center, $r_k \leq o_k$

L_i Priority of i^{th} crisis, $i = 1, 2, ... 10$ where 1 is the lowest and 10 is the highest level

T Set of response times of resources from resource center to crises, where $T = \{t_{i,j}\}$

$t_{i,j}$ Time to by j^{th} resource center to reach i^{th} crisis

NS_i Set of strategies constituting the Nash Equilibrium solution

CHAPTER 4

OPTIMAL ALLOCATION MODEL USING NASH EQUILIBRIUM

The optimal allocation of resources in the crisis scenario is dependent on the modeling of the problem. In the previous chapter, we observed the transformation of such a scenario into a noncooperative game as it provides a framework for analyzing strategic interactions. In this chapter, the details of this transformation are presented. We propose a definition of a strategy in the context of this thesis and elaborate the process of the formulation and pruning of the strategy space. The basic idea behind this approach is to apply heuristics to eliminate strategies which contribute to infeasible and "poor" solutions. Another goal is to control the explosion of the strategy space as the dimension of the problem increases. The computation of a Nash Equilibrium increases in complexity as the strategy space grows and hence it becomes necessary to prune the strategy space. After the strategy spaces are constructed, we play a noncooperative strategic game and obtain a probability distribution over the strategy set of each crisis which is used to determine the allocations.

4.1 Structure of a Strategy

A game is modeled around its information set and a strategy captures the essence of this information in formulating an "action" for a player. Each player can play the game by selecting an action from its own set of "actions" or strategies, S_i. The definition of an "action" or a strategy

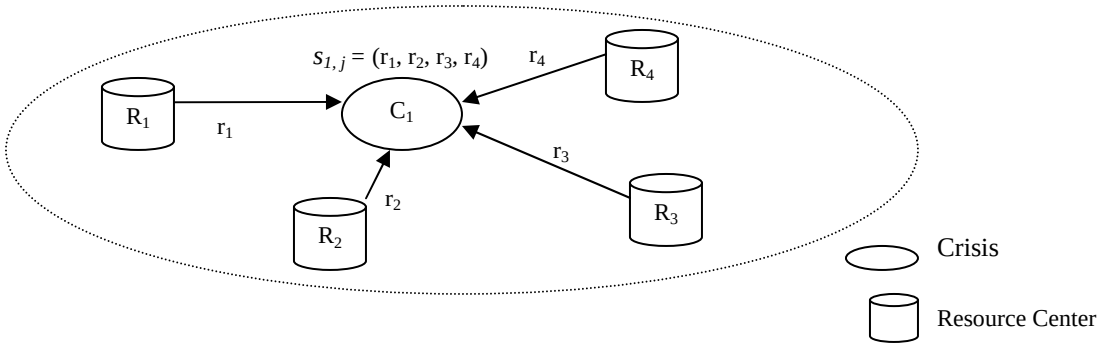


Fig 4.1 Formulation of a Strategy

plays a significant role in determining the outcome of any game. In our crisis scenario model, the information available consists of,

- Number of crisis locations,
- Number of resource centers,
- Time taken to travel from the resource centers to each crisis,
- Priority of each crisis
- Number of resources requested by each crisis

The crises are modeled as players and the amount of resources allocated to them constitutes a strategy. Let $C = \{C_i\}$ for $i \leq n$, be a finite set of crises and $R = \{R_j\}$ for $j \leq m$ be a finite set of resource centers, where $n \in N$ is the number of crises and $m \in N$ is the number of resource centers. Let S_i be the finite set of pure strategies available to crisis $i \in C$. In the context of this work, we define a strategy as an n-tuple consisting of non-negative integers, one for each of the m resource centers. We write,

$$s_{i,j} = (r_1, r_2, ..r_k ... r_m) \quad \forall r_k \in N \text{ and } s_{i,j} \in S_i$$

where $s_{i,j}$ denotes the j^{th} strategy of the i^{th} crisis and r_k is number of resources contributed by resource center R_k from its pool of resources.

4.2 Generation of Strategies

The concept of a noncooperative, normal form game is used in this work. All the players move simultaneously in a game which can be represented in a matrix form. Hence, it becomes necessary to enumerate all possible and feasible allocation strategies before the game is played. With each resource center equipped with r_k resources, there are $o_1*o_2*\ldots o_k*\ldots o_m$ different possibilities of allocating resources from the different centers to a crisis. Every crisis has its set of allocation strategies,

$$S_i = \{s_1, s_2, \ldots s_{g(i)}\} \quad \text{where} \quad g(i) \le o_1*o_2*\ldots o_k*\ldots o_m,\ g(i) \in N,\ i = 1,2,\ldots,n$$

We use a recursive algorithm to generate the combinations of allocations that each crisis can select from. This is repeated for each of the crises. Figure 3.4 shows the algorithm used to generate these combinations. During the process of generating the strategies, we apply certain constraints on them to eliminate infeasible strategies.

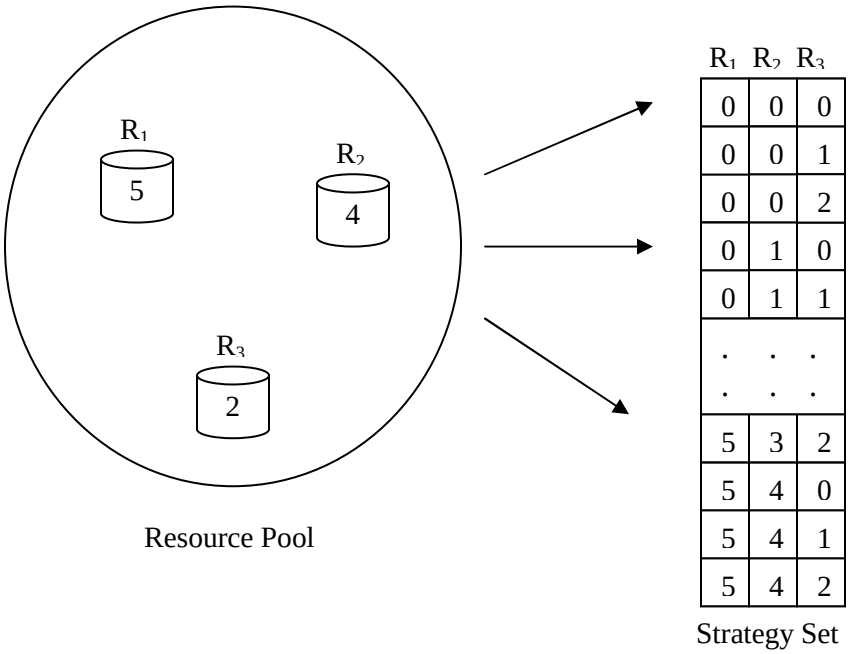


Fig 4.2 Generation of Strategies

Algorithm: GenStrat - Generation of strategies for each crisis
Input: Resource array containing number of resource units in each center - *R[m]*, Number of resource centers *m*, Crisis array containing requests of each crisis – *C[n]*,Number of crises *n*
Output: Set of strategies S_i for each crisis

Declare *T[m]* ← Temporary array
Declare *head* ← Pointer to ordered list of strategies

for crises in C_i , $i \leftarrow 1$ to *n*
Strategy Set $S_i \leftarrow$ Call RecStrat (*R, m, C, n, T, 0, head*)
end

Fig 4.3 Algorithm to Generate Strategies

4.2.1 Pruning of Strategies

Since this is a game of complete information, the number of resources requested by a crisis location is known. During the allocation of a strategy to a crisis, it is imperative that a crisis is not allocated more resources than it needs. This is to avoid unfair allocation and wastage of resources. Resources that were allocated in excess of the requirement could have been used for crises whose needs are more urgent or for other crises which might occur.

Let $Q = \{q_i\}$ be the set of requirements of all the crises where $i = 1, 2, \ldots n$ and $q_i \in N$. Similarly, let $O = \{o_j\}$ be the set of the resources available at the resource centers where $j = 1, 2, \ldots, m$ and $o_j \in N$. We apply our first constraint to the strategy generation process.

For any strategy, $s_{i,j} \in S_i$, $\sum r_k \leq q_i$ where $i = 1, 2, \ldots, n$ and $k = 1, 2, \ldots, m$ ---------- (1)

The above statement implies that in any strategy $s_{i,j}$ belonging to crisis C_i, the sum of all the entries in the strategy tuple should not exceed the requirement q_i of that crisis. For e.g., if crisis C_1 requires 4 resources and it has two strategies – $s_{1,1}(1,2,1)$ and $s_{1,2}(1,2,2)$, the strategy $s_{1,2}$, 2nd strategy of crisis C_1, is invalid as 1+2+2 = 5 > 4.

While generating strategies, it should be kept in mind that the individual entries in a strategy tuple should not exceed the availability of the corresponding resource center. Hence, we arrive at the second constraint on the strategies.

For any strategy, $s_{i,j} \in S_i$, $r_k \leq o_k$ where $i = 1, 2, \ldots, n$ and $k = 1, 2, \ldots, m$ ---------- (2)

The above condition indicates that in any strategy $s_{i,j}$ belonging to crisis C_i, the individual entries in the strategy tuple i.e. $s_{i,j} = (r_1, r_2, r_3, \ldots r_k, \ldots, r_m)$, should not exceed the corresponding resource center's total capacity i.e. $r_1 \leq p_1$, $r_2 \leq p_2$, $r_3 \leq p_3$,… $r_k \leq p_k$,…, $r_m \leq p_m$. For e.g., Let there be three resource centers R_1, R_2 and R_3, with capacities 3, 4 and 2 units respectively. A strategy $s_{i,j} =$ (4, 3, 2) (j^{th} strategy of i^{th} crisis) is invalid as the first entry in the tuple corresponding to the contribution of resource center R_1 is 4 whereas the capacity of R_1 is 3.

Algorithm: RecursiveStrat - Generation of strategies for each crisis
Input: Resource array containing number of resource units in each center - *R[m]*, Number of resource centers *m*, Crisis array containing requests of each crisis – *C[n]*,Number of crises *n*, Temporary array *T[m]*, Variables *curr*, Pointer to ordered list of strategies *head*
Output: Strategy set S_i

```
temp = curr + 1
if temp == m then
        last = 1;
else
        last = 0;
end

for resource units in R_i , i ← 1 to curr
        T[curr] = i;
        if last == 1 then
                head ← Call AddStrat
        else
                head ← Call RecursiveStrat (R, m, C, n, T, curr+1, head)
        end
end
return head
```

Fig 4.4 Recursive Algorithm to Generate Strategy Set for each Crisis

4.2.2 Cost Function

In modeling this problem, we provide the user with an option to select n best strategies from each crisis's set and play the game. These strategies are ordered in ascending order of cost to the crisis. Here cost is not tangible and is used as a measure of *practicality*. The objective

behind the ordering is to sort the strategies by those that provide *the greatest number of resources in the shortest possible time*. In a strategy, we compare the time taken to reach a crisis from each resource center and the number of resources contributed by that resource center as the factors in determining the cost of that strategy.

Let $T = \{t_{i,j}\}$ be the set of measures of time taken to reach each of the crises from the resource centers. $t_{i,j}$ is the time taken to reach the i^{th} crisis from the j^{th} resource center. Here $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$ where n is the number of crises and m is the number of resource centers. For any strategy,

$$\text{Cost}(s_{i,j}) = \min\left\{\frac{t_{i,1}}{r_1}, \frac{t_{i,2}}{r_2}, \ldots\ldots, \frac{t_{i,m}}{r_m}\right\} \text{ for any strategy } s_{i,j} = (r_1, r_2, ..r_k ... r_m) \quad \ldots\ldots\ldots (3)$$

Cost ($s_{i,j}$) measures the ratio between the time taken to reach a crisis i from each of the resource centers in a strategy tuple $s_{i,j}$ and the number of units contributed by each of the resource centers.

It is not uncommon for two strategies to have the same cost. In such a case, we examine the two strategies and look for the next lowest value of the ratio in the tuples. The strategy with the next lower value is ranked higher among the two strategies being compared. Let Cost ($s_{i,j}$) = $\min\left\{\frac{t_{i,1}}{r_1}, \frac{t_{i,2}}{r_2}, \ldots\ldots, \frac{t_{i,m}}{r_m}\right\} = a$ where $a = \frac{t_{i,p}}{r_p}$ and Cost ($s_{i,k}$) = $\min\left\{\frac{t_{i,1}}{r_1}, \frac{t_{i,2}}{r_2}, \ldots\ldots, \frac{t_{i,m}}{r_m}\right\} = b$

where $b = \frac{t_{i,q}}{r_q}$ where ($s_{i,j}$) and ($s_{i,k}$) are j^{th} and k^{th} strategies of crisis i.

If $a = b$, then compare $\min\left\{\frac{t_{i,1}}{r_1}, \frac{t_{i,2}}{r_2}, \ldots \frac{t_{i,p-1}}{r_{p-1}}, \frac{t_{i,p+1}}{r_{p+1}} \ldots, \frac{t_{i,m}}{r_m}\right\}$, and

$$\min\left\{\frac{t_{i,1}}{r_1}, \frac{t_{i,2}}{r_2}, \ldots \frac{t_{i,q-1}}{r_{q-1}}, \frac{t_{i,q+1}}{r_{q+1}} \ldots, \frac{t_{i,m}}{r_m}\right\} \quad \ldots\ldots\ldots\ldots\ldots (4)$$

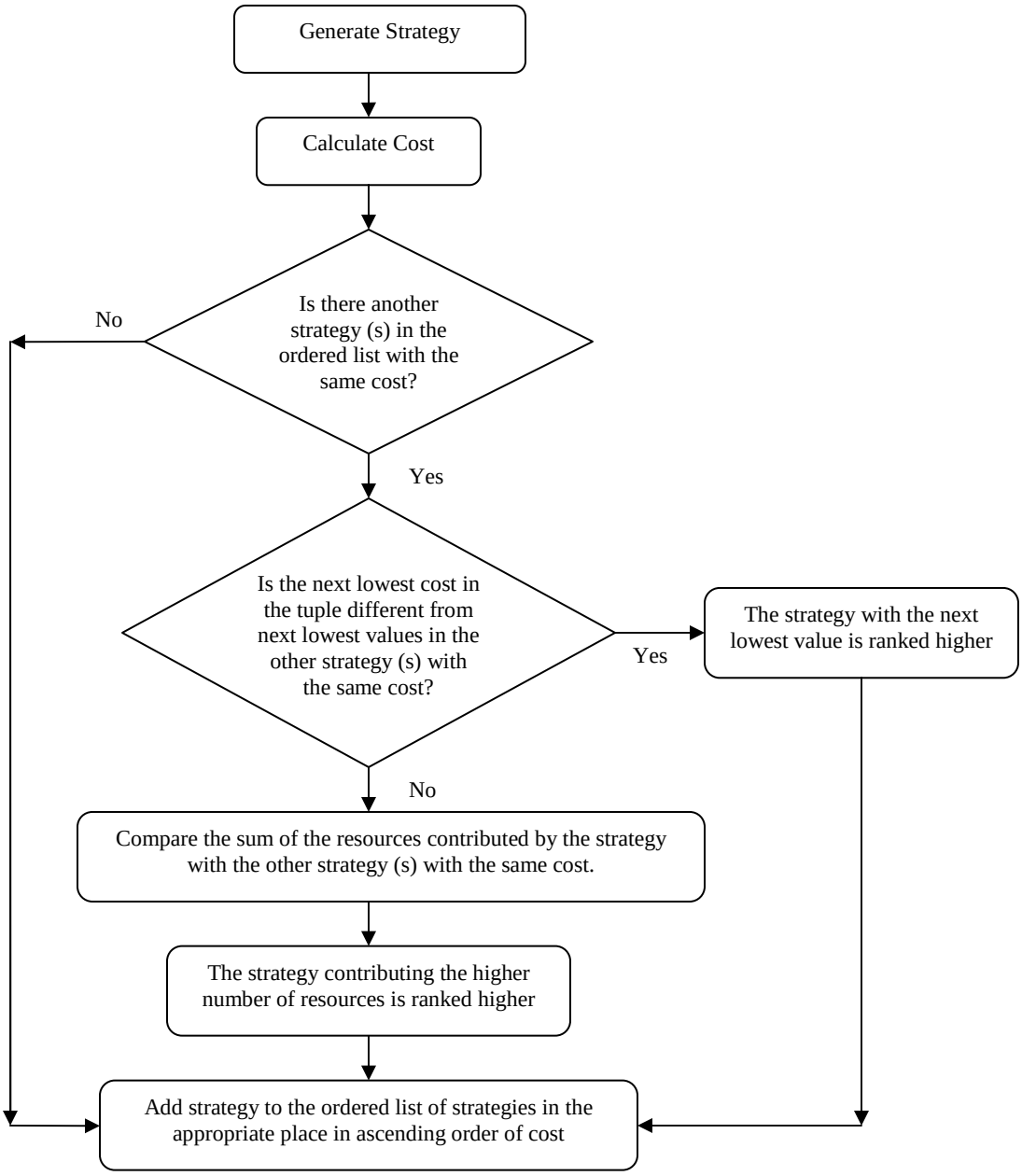


Fig 4.5 Flowchart Illustrating Ordering of Strategies

Algorithm: AddStrat - Add strategy to strategy set of each crisis
Input: Strategy *S[m]*, Number of resource centers *m*, Pointer to ordered list of strategies *head*, Crisis array containing requests of each crisis – *C[n]*,Number of crises *n*
Output: Pointer to ordered list of strategies head

Declare *sum* ← Temporary Variable

for resource units in S_i , $i \leftarrow 1$ to *m*
    $sum \leftarrow sum + S_i$
end

if sum $\leq$ C[present crisis] **then**
    Compute *cost* of strategy *S*
    Determine order strategy in list of strategies based on *cost*
    Add strategy to strategy set
end
return *head*

Fig 4.6 Algorithm to Add a Strategy to a Strategy Set

The process above (4) is repeated until we find two unequal costs. If two strategies being compared have identical $\frac{t_{i,k}}{r_k}$ values in their tuples, we compare the two strategies for the one which contributes the higher sum total of resources to the crisis. The strategy contributing the higher number of resources is ranked higher among the two strategies being compared,

i.e. compare $\sum r_k$ where $k = 1, 2, ..., m$ in $(s_{i,j})$ and $(s_{i,k})$ for the higher value ..... (5)

We perform the ordering of the strategies at the time of generation to avoid the additional overhead of sorting a huge list of strategies at the end thereby adding the time taken to sort strategies of each of the crises to the overall computation time. Figure 3.5 shows the process of ordering of strategies.

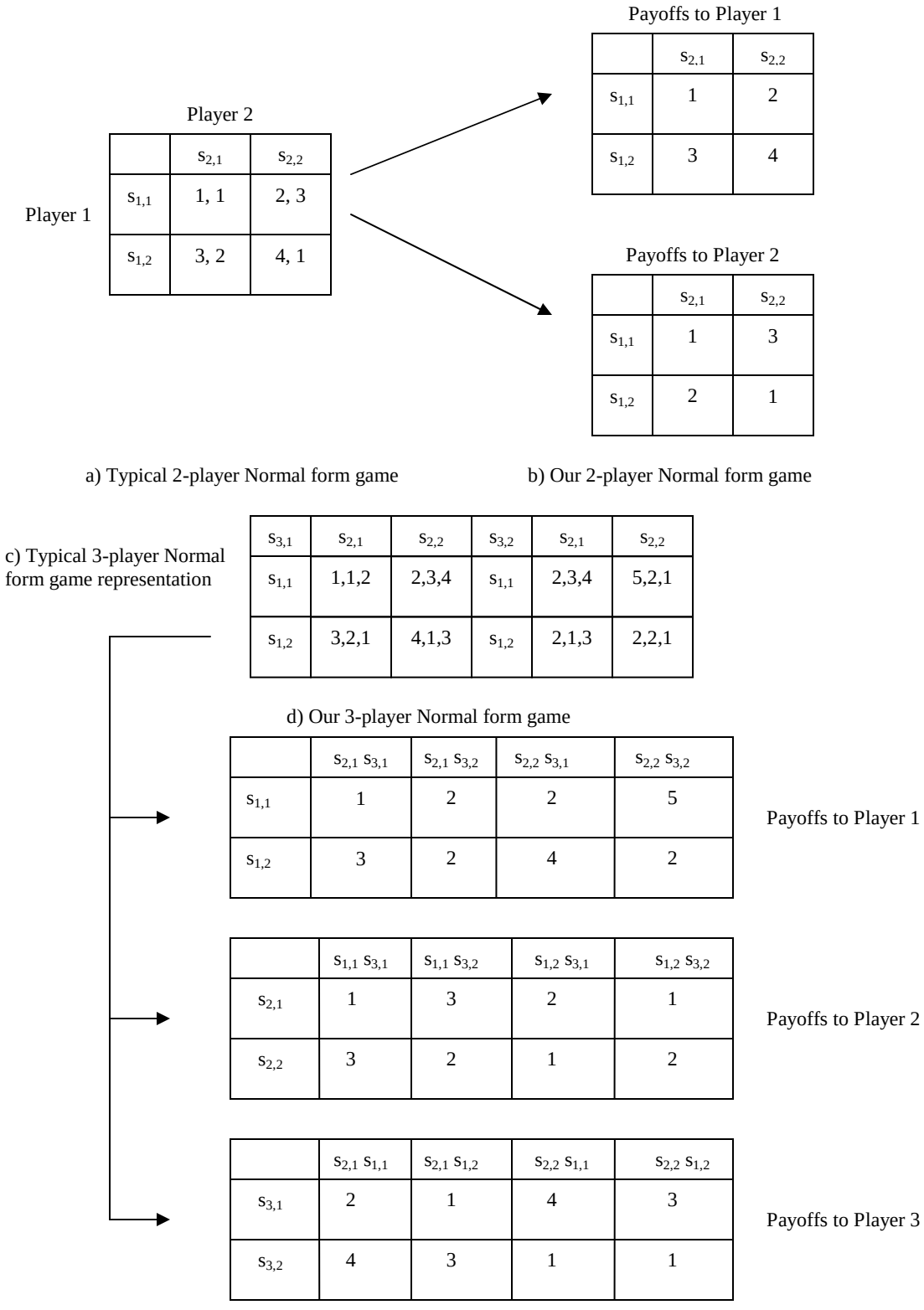


Figure 4.7 Normal Form Game Representation

4.3 Payoff Modeling

A normal form representation of an *n*-player game is specified with,

- A finite set of *n* players,
- A finite set of strategies $S = \{S_1, S_2, ..., S_n\}$ for each player
- A utility/payoff function u_i to be applied on the set *S*

Figure 3.7 shows the typical representation and our representation of a 2-player game. In (a), the strategies of player 1 are vertical and the strategies of player 2 are horizontal. Each cell has two entries – the first entry is the payoff to player 1 and the second entry is the payoff to player 2 when player 1 and player 2 choose the strategies at the far left and top respectively. Similarly, in (c) the matrix is divided into two parts. Each part is similar to (a) but represents a payoff when a third player chooses its first strategy (left half) and its second strategy (second half). In (b) and (d), we divide (a) into two matrices and (c) into three matrices respectively, one for each player's payoffs for choosing a particular combination.

4.3.1 Creation of Payoff Matrices

A normal form game can be represented as a matrix. For the purpose of our implementation we create *n* payoff matrices, one for each crisis. In our representation of the payoff matrices in Figure 3.8, the first column of each crisis's matrix holds its strategies, one on each row starting from the second row. The first row of the crisis's matrix holds the combinations of strategies selected by all the other crises, one on each column starting from the second column. The 'X' in the first matrix is for the payoff for the first crisis when C_1 selects $s_{1,1}$ and C_2 selects $s_{2,1}$ and C_3 selects $s_{3,1}$.

Strategy Sets for Crises – C_1, C_2 and C_3

Crisis C_1	Crisis C_2	Crisis C_3
$s_{1,1}$	$s_{2,1}$	$s_{3,1}$
$s_{1,2}$	$s_{2,2}$	$s_{3,2}$
$s_{1,3}$	$s_{2,3}$	$s_{3,3}$

Payoff matrix of crisis C_1

	$s_{2,1}$ $s_{3,1}$	$s_{2,1}$ $s_{3,2}$	$s_{2,1}$ $s_{3,3}$	$s_{2,2}$ $s_{3,1}$	$s_{2,2}$ $s_{3,2}$	$s_{2,2}$ $s_{3,3}$	$s_{2,3}$ $s_{3,1}$	$s_{2,3}$ $s_{3,2}$	$s_{2,3}$ $s_{3,3}$
$s_{1,1}$	X								
$s_{1,2}$									
$s_{1,3}$									

Payoff matrix of crisis C_2

	$s_{1,1}$ $s_{3,1}$	$s_{1,1}$ $s_{3,2}$	$s_{1,1}$ $s_{3,3}$	$s_{1,2}$ $s_{3,1}$	$s_{1,2}$ $s_{3,2}$	$s_{1,2}$ $s_{3,3}$	$s_{1,3}$ $s_{3,1}$	$s_{1,3}$ $s_{3,2}$	$s_{1,3}$ $s_{3,3}$
$s_{2,1}$									
$s_{2,2}$									
$s_{2,3}$									

Payoff matrix of crisis C_3

	$s_{1,1}$ $s_{2,1}$	$s_{1,1}$ $s_{2,2}$	$s_{1,1}$ $s_{2,3}$	$s_{1,2}$ $s_{2,1}$	$s_{1,2}$ $s_{2,2}$	$s_{1,2}$ $s_{2,3}$	$s_{1,3}$ $s_{2,1}$	$s_{1,3}$ $s_{2,2}$	$s_{1,3}$ $s_{2,3}$
$s_{3,1}$									
$s_{3,2}$									
$s_{3,3}$									

Fig 4.8 Structure of Payoff Matrices for Crises

4.3.2 The Payoff Function

In any game, the best action for any player depends on the actions of the other players. A payoff to a crisis for choosing a particular strategy when the other crises make their selection can be represented as a gain to the crisis or a loss to the other crises. In our model, we depict it as a summation of the losses to the other players and each player tries to maximize this loss to other players. The payoff to a crisis is representative of the loss incurred by the other crises on the allocation of a particular strategy to the crisis in question and the remaining crises are allocated their strategies. Essentially, the possible combinations of strategies remain the same. Each matrix captures the payoff to a particular crisis for a particular combination.

In our model, every crisis is assigned a priority L on a scale of 1 to 10 to indicate its severity. This priority is used as a weight in a payoff function to facilitate the calculation of the loss to a crisis(s).

Payoff to k^{th} strategy of i^{th} crisis when the crises $t \neq i$ choose $s_{m \neq i,\, l} \in \{S_1, S_2 \ldots S_{i-1}, Si_{+1}, Sn\}$ is given by,

$$\left(\sum_{t=1}^{n} \sum_{j=1}^{m} \left(\left(r_{j,St \neq i,l} / \left(o_j - r_{j,St=i,k} \right) \right) - \left(r_{j,St \neq i,l} / o_j \right) \right) * L_{t \neq i} \right) - constant$$

$r_{j,St \neq i,l}$ Resources contributed by the j^{th} resource center of l^{th} strategy of crisis $C_{t \neq i}$

$r_{j,St=i,k}$ Resources contributed by the j^{th} resource center of k^{th} strategy of crisis $C_{t=i}$

o_j Total number of resources available at resource center R_j

$L_{t \neq i}$ Priority of crisis $C_{t \neq i}$

M Number of resource centers

n Number of crises

The term $\left(r_{j,St \neq i,l} / \left(o_j - r_{j,St=i,k} \right) \right)$ refers to the ratio between the resources contributed by a strategy and number of resources available from that resource center after allocating to the crisis in question, C_i. The term $\left(r_{j,St \neq i,l} / o_j \right)$ refers to the ratio between the resources contributed by a strategy and number of resources available from that resource center without allocating to the crisis in question, C_i. The difference captures the *loss* to the crises $C_{t \neq i}$ and the priority of the crises adds a weight to the loss. The last term of the expression, *constant*, is value which is either 0 or ∞. If the combination of strategies is feasible, then *constant* = 0, else *constant* = ∞ (or a very high number).

Consider a combination of strategies, $\{s_{1,\, p}, s_{2,\, q}, \ldots., s_{n,\, r}\}$. Let $\{r_{\sum 1,} r_{\sum 2,\, \ldots,} r_{\sum m}\}$ be the sum of resources.

$(r_1, r_2, ..., r_m)$ belonging to $s_{1,p}$

\+ $(r_1, r_2, ..., r_m)$ belonging to $s_{2,q}$

\+ ....

\+ $(r_1, r_2, ..., r_m)$ belonging to $s_{n,r}$

_____

$(r_{\sum 1}, r_{\sum 2, ..., } r_{\sum m})$

If $r_{\sum i} \geq o_i$ for *any* $i = 1, 2, ..., m$, the combination of strategies becomes infeasible.

4.4 Algorithm to Approximate Nash Equilibrium

We apply a variant of the Scarf-Hansen fixed-point algorithm [69, 70] to approximate a Nash Equilibrium point in our noncooperative game. The algorithm is based on a combinatorial theorem [69] which is expressed in terms of a primitive set. Consider a collection of h-dimensional vectors $X = (x^{1,} x^{2,} ...., x^{h})$ of the form $(m_1/D, ..., m_h/D)$ with each value greater than or equal to -1 and summing up to D which is a very large number, typically a multiple of the number of crises. Let the numerators of the vectors in X be [69],

$$M = \begin{matrix} m_{11} & m_{12} & .... & m_{1h} \\ m_{21} & m_{22} & .... & m_{2h} \\ . & . & & . \\ . & . & & . \\ m_{h1} & m_{h2} & .... & m_{hh} \end{matrix}$$

The above matrix is a primitive set *if and only if* and there is a rearrangement of the columns and a permutation of the labels of the columns, $I(l)$ such that[69],

1. The lth column is identical to column l-1, except for the two rows $I(l)$-1 and $I(l)$

2. $m_{k,l} = m_{k,l-1} + 1$ for k = $I(l)$-1

$m_{k,l} = m_{k,l-1} - 1$ for k = $I(l)$

Note: For $l = 1$, l-1 = h. Similarly, $I(l) = 1$, $I(l) - 1 = h$

Fig 3.9 shows the steps of the algorithm used to compute the Nash Equilibrium solution. As shown in the figure the output is a probability vector $p = (p_1, p_2, \ldots p_i \ldots p_n)$. p_i is a probability distribution over the strategy set S_i of crisis C_i.

$p_i = \{p_{i,j}\}$ where j ≤ n and i ≤ g_j, total number of strategies in S_j

Algorithm: NashSolve - Approximate Nash Equilibrium
Input: n – number of crises, X, Payoff matrices
Output: Probability vector p

Calculate h = Total number of strategies of all crises – n +1

While label of $x^j \neq 1$

Let x^j be an arbitrary nonnegative vector from X. We associate it with the probability vectors $(p_1^j, p_2^j, \ldots p_n^j)$ as follows:

$t_0 = 0$ and $t_i = s_{i-1}$ where $i = 1, 2, \ldots, n$ and s_i is the number of strategies of crisis Ci

$$r_{ki} = 1 + \sum_{v=0}^{i-1} t_v + k \qquad \text{where } k = 1, \ldots, t_i \text{ and } i = 1, \ldots, n$$

$$p_{k,i}^j = n \cdot x_{r_{ki}}^j \qquad \text{where } k = 1, \ldots, t_i \text{ and } i = 1, \ldots, n$$

$$p_{s_i,i}^j = 1 - \sum_{i=1}^{t_i} p_{k,i}^j \qquad \text{where } p_{k,i}^j \geq 0$$

if $p_{s_i,i}^j \geq 0$ **then**

Let $B_{ki} = B_{ki}(p_1^j, .., p_{i-1}^j, p_{i+1}^j \ldots p_n^j)$ be the expected payoff to a crisis it uses its k^{th} strategy and k_i is the lowest index for which

$B_{k_i i}(p_1^j, .., p_{i-1}^j, p_{i+1}^j \ldots p_n^j) \geq B_{ki}(p_1^j, .., p_{i-1}^j, p_{i+1}^j \ldots p_n^j)$

The vector x^j is labeled $1 + \sum_{u=0}^{i-1} t_u + k_i$, where $i = \min\{ l \mid p_{s_l,l}^j > 0 \text{ and } k_l \neq s_i \}$

Perform replacement step

end

if $p_{s_i,i}^j < 0$ **or** $p_{s_i,i}^j = 0$ for all i **or** $k_i = s_i$ for all i with $p_{s_i,i}^j > 0$ **then**

x^j is labeled 1

Terminate algorithm

end

end

Fig 4.9 Algorithm to Approximate Nash Equilibrium

We compute $\max_j (p_{i,j})$ for each i

The Nash Equilibrium solution is given by,

$$NS_i = \left\{ s_{i,\max_j (p_{i,j})} \right\}$$

$\max_j (p_{i,j})$ corresponds to the strategy with maximum value of probability among strategies of a crisis. In the event that there are two strategies with identical probabilities, we pick the one with the lower cost to the crisis.

4.5 Software Implementation

Crisis management encompasses a whole range of activities from "sensing" a crisis to deployment of resources to monitoring of crisis development. The objective of our work is to provide an automated mechanism for determining the number of units assigned to each crisis location based on priority and requirement. In this work, we have implemented a tool that

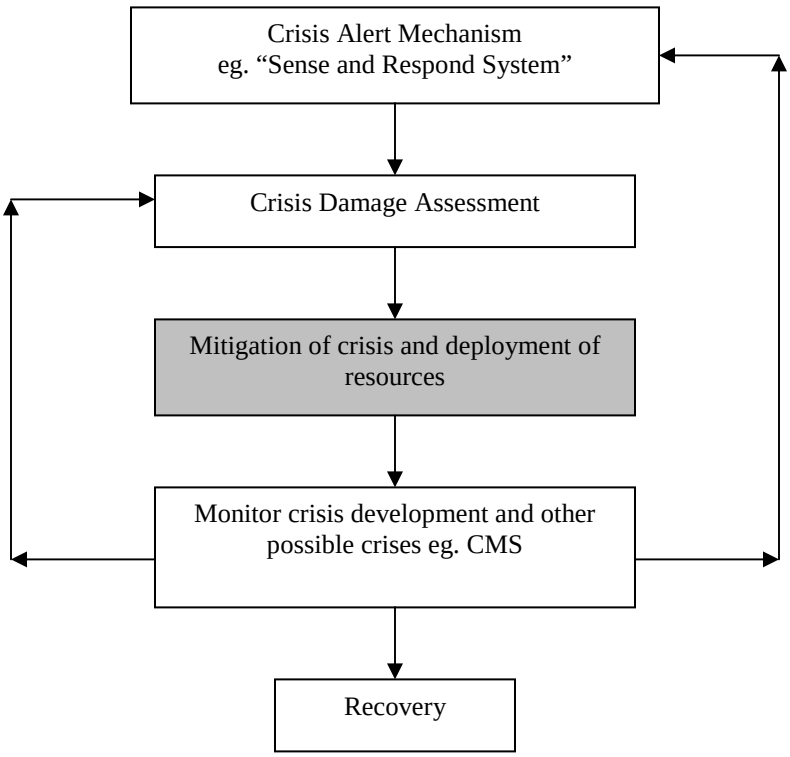


Fig 4.10 Overview of Crisis Management System

determines an optimal allocation of resources in a multi-crisis environment. The user provides a set of inputs to the system, namely, resource capacities, crisis requests, crisis priorities and response time information. The software evaluates the information using the underlying algorithm and presents the user with an optimal allocation of resources from each center. In Fig 4.11, the grey box indicates where our tool for allocation of resources to crises will fit into a crisis management model.

4.5.1 System Input and Output

Every system is unique in the nature of inputs fed into it and the output presented to the user. Below are the specifications of the input and the output of our system:

Input:

- Number of crises (2 - 5 crises)
- Number of resource centers (2 – 10 centers)
- Resources requested by each crisis (8 – 15 resources)
- Resources available at each resource center (2 – 10 resources)
- Priority of each crisis (1 – 10; 1 being the lowest and 10 being the highest)
- Time taken to travel between crisis and resource center (in minutes)

Output:

Allocation consisting of combination of resources contributed by each resource center

4.5.2 Object-Oriented Design

We have used an object oriented design to implement our solution because of the two main benefits:

Maintainability – This is achieved through a simplified mapping to the problem domain, which results in less analysis effort, less complexity in system design and easier user verification.

Reusability – Segments of the structured code can be reused by adding new functionalities with slight or no modification. This reduces implementation time, localizes the modifications in code when a change in implementation is required and also increases the possibility that prior testing has removed bugs.

The programming language used to implement it is C++. It was chosen because of its ability to program in a C-like style, or an object-oriented style or both and also its ability to utilize the predefined classes and be able to create user-defined classes to characterize the features of the input. Object oriented design allows us to organize data into discrete, distinguishable entities called objects. A single object has a state and behavior associated with it. For example, in our work, crises and resource centers are objects. Each has its own characteristics and behavior. A crisis is described in terms of it severity, requests and location. A resource center is described in terms of its capacity. Furthermore, each object has its own characteristic behavior. For example, each crisis generates its strategy set.

4.5.3 Overview of Classes and Functions

We have the following four classes with object functions in our implementation:

Table 4.1 Overview of Classes Used

Class	Attributes	Behavior
Resource Center	Name, Capacity	-
Crisis	Name, Priority, Number of Strategies, Pointer to Strategy Set, Response time from each Resource center	• Recursively generates all possible allocations from resource centers • Adds a strategy to its set • Sorts its strategies based on cost
Payoff Matrix	Name, Number of Crises, Number of combinations, Structure for each Combination	• Generates rows and columns of a matrix • Computes payoffs
Nash	Primitive Set, X Vector, Probability Vector, D, n, k array and t array	• Computes the Nash Equilibrium

The following functions have been used:

RecNum()

- Class : Crisis
- Input : Number of resource centers, resource center capacities
- Function : Generate strategies for a crisis from the available resources from each center

AddStrat()

- Class : Crisis

- Input : Strategy of a crisis
- Function : Add strategy to a crisis' strategy set

costSort()

- Class : Crisis
- Input : Strategy of a crisis
- Function : Locate the ordering of a strategy based on its cost

compareStrategy()

- Class : Crisis
- Input : Strategies having the same cost
- Function : Determine ordering of two strategies with same cost

GenerateComIndex()

- Class : PayoffMatrix
- Input : Number of crises
- Function : Generate indexes for combination of strategies belonging to different crises

generateComIndex()

- Class : PayoffMatrix
- Input : Number of crises
- Function : Generate indexes for combination of strategies belonging to different crises

getRowStrategy()

- Class : PayoffMatrix
- Input : Number of crises

- Function : Generate strategies in each row of payoff matrix

getColumnStrategy()

- Class : PayoffMatrix
- Input : Number of crises, Strategy set of each crisis
- Function : Generate combinations of strategies in each column of payoff matrix

getPayoff()

- Class : PayoffMatrix
- Input : Number of crises
- Function : Compute payoffs in the payoff matrix of each crisis

selectedStrategy()

- Class : Nash
- Input : Crises, resource centers, payoff matrices
- Function : Determine the optimal allocation for each crisis

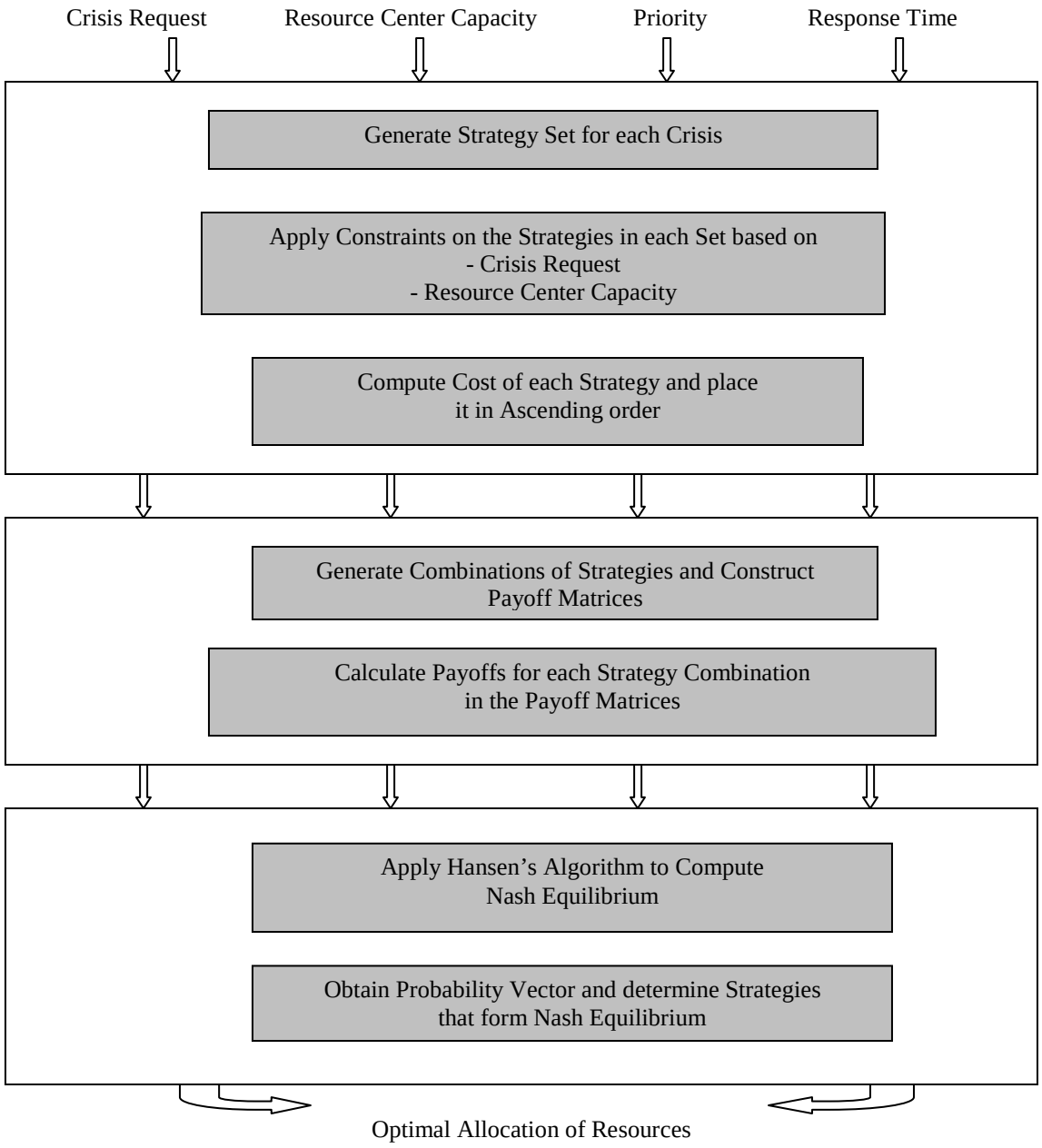


Fig 4.11 Workflow Model of the Proposed System

CHAPTER 5

EXPERIMENTAL RESULTS

This chapter presents a set of experimental results to verify the efficacy of the solution obtained using our implementation. The implementation has been divided into three phases – generation of strategies, computation of payoff matrices and finally, the generation of the Nash Equilibrium solution. For the integration of the three phases, the output of each phase is made compatible for the next step. The user inputs information regarding number of crises, requests, response times and criticality levels. C++ is the programming language chosen for implementing our tool to obtain an optimal solution. The object-oriented characteristic of C++ facilitates modeling of the characteristics of players and strategies, and the modularization of the implementation.

In this chapter, we observe the performance of our approach in various crisis scenarios by varying the inputs and noting its effects on execution time and quality of the solution. It is logical to assume that as the dimensionality of an input set increases, so does the execution time of an implementation. In our implementation, the process of generating strategies and the computation a Nash Equilibrium are time intensive tasks and we examine the effect of increasing number of crises on the execution time. Also, the total availability of resources in a resource pool versus the total demand or requests made by crises are contributing factors in determining the quality of the solution.

We have performed experiments on various test cases by altering inputs such as crisis requests and resource center capacities. The test cases are used to study the effects of inputs on the time taken to reach a solution and also to determine the quality of a solution. Furthermore, we

apply the fairness measures as described in [79] to quantify the feasibility and fairness of the implementation. All experiments were run on a Sun Fire V880 UltraSPARC III server that features eight 900 MHz processors and 32 GB of memory running the SunOS.

5.1 Fairness

Typically, in a priority based system, a high priority event would receive its request and the lower priority event would receive less than its fair share and possibly starve due to lack of resources. This is not very different from a first-come-first-serve principle where the more critical event is sent ahead in line to claim its share of resources. Although, it would seem logical to allow a more critical event to satisfy its request, such a scheme suffers from a high degree of unfairness. A priority-based system would perform reasonably well when there is sufficient number of resources in the resource pool. However, the problem arises when the total demand exceeds the total supply. Higher priority crises satisfy their requests, while lower priority crises events suffer from starvation due to inevitable shortage. In a crisis environment, this is unacceptable as starvation of a lesser priority crisis could lead to further loss in life and property and increase the possibility of worsening of a crisis scenario.

Generally, a system is deemed to be fair or unfair based on whether or not it meets certain requirements or not. Generally, a system is considered fair if it meets certain criteria on throughput or delay, and it is considered unfair if these criteria are not met. For example, a scheduling algorithm is fair or unfair depending on whether any user receives a throughput of x bits/sec or not. In our system, fairness would be determined by whether or not a crisis receives its share of resources. We use the fairness concepts of [79] to determine the fairness of our implementation. In our approach, we compare the best-cost strategy of a crisis with the actual-cost strategy, i.e. the cost of the strategy that is assigned as the Nash Equilibrium solution. A measure of fairness of the system can be measured in terms of the self-fairness of the individual crises. The proportion of the cost associated with crisis *j* that is deemed to be fair is given by,

$$p_{fair,j} = \frac{\cos t_{j,lowest}}{\sum_{i=1}^{i} \cos t_{i,lowest}}$$

$$= \frac{r_j}{r_T}$$

Here, $\cos t_{j,lowest}$ is the lowest or best cost strategy of a crisis. In Chapter 4, we have described the ordering of strategies based on cost. We have used the term r_j for brevity.

$$p_{actual,j} = \frac{\cos t_{j,actual}}{\sum_{i=1}^{n} \cos t_{i,actual}}$$

Here, $\cos t_{j,actual}$ is the cost of the strategy assigned to a crisis as part of the Nash Solution.

The definition of the self-fairness of a given user is,

$$f_j = \frac{\log\left(p_{actual,j}\right)}{\log\left(r_j / r_T\right)}$$

We also define,

$$C_k = \frac{1 + \frac{1}{\log\left(r_1 / r_T\right)}}{1 + \frac{1}{\log\left(r_k / r_T\right)}}$$

The values C_k are normalization constants. Their objective is to ensure that the maximum value of the weighted average fairness is unity and the maximum occurs when each user consumes its fair share of the resources. The average fairness of the system is given by,

$$F = \frac{r_T \sum_{k=1}^{n} C_k p_k f_k}{\sum_{k=1}^{n} C_k r_k}$$

The average fairness value of F ranges between 0 and 1. The value of unity results when a crisis consumes resources using a strategy with the lowest possible cost.

Tables 5.1 (a-d) show the fairness measure for test cases with varying differences between demand and supply. Every crisis attempts to select a strategy from its set of strategies that costs the least. We observe a slight decrease in fairness as the total request for resources from all the crises exceeds the total resources available from all the centers.

Table 5.1 Fairness Measures

Demand vs. Supply	Degree of Fairness
Total demand 21%-50% less than total supply	0.837309
Total demand 1%-20% less than total supply	0.827117
Total demand = Total availability	0.833342
Total demand 1%-20% greater than total supply	0.803526
Total demand 21%-50% greater than total supply	0.805155

(a) Crisis Scenario with 2 Crises

Table 5.1 Continued

Demand vs. Supply	Degree of Fairness
Total demand 21%-50% less than total supply	0.811523
Total demand 1%-20% less than total supply	0.848788
Total demand = Total availability	0.891354
Total demand 1%-20% greater than total supply	0.71144
Total demand 21%-50% greater than total supply	0.781591

(b) Crisis Scenario with 3 Crises

Demand vs. Supply	Degree of Fairness
Total demand 21%-50% less than total supply	0.822978
Total demand 1%-20% less than total supply	0.818603
Total demand = Total availability	0.845423
Total demand 1%-20% greater than total supply	0.710171
Total demand 21%-50% greater than total supply	0.691537

(c) Crisis Scenario with 4 Crises

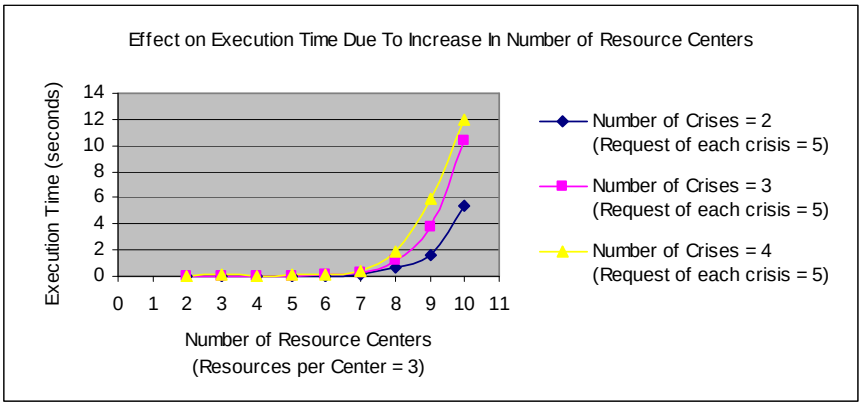
Demand vs. Supply	Degree of Fairness
Total demand 21%-50% less than total supply	0.789292
Total demand 1%-20% less than total supply	0.769231
Total demand = Total availability	0.769231
Total demand 1%-20% greater than total supply	0.769858
Total demand 21%-50% greater than total supply	0.769231

(d) Crisis Scenario with 5 Crises

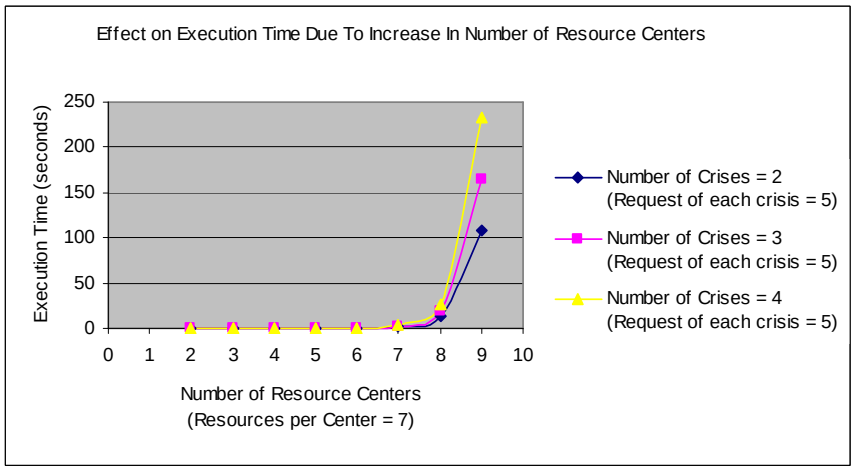
5.2 Execution Time

Although the optimality of the solution is important, the time taken to arrive at an optimal solution is an important factor in determining the feasibility of our approach. For example, on the occurrence of 3-4 crises simultaneously, a formulation which took more than 1-3 minutes would not be tolerated. Every second utilized to construct a solution has a direct effect on the response time of the emergency units and consequently on their ability to prevent damage. The nature of the problem is such that we are limited in scope in terms of comparison with other works. We analyze the implementation by varying the inputs and observing its effects on the runtime.

The graphs in Fig 5.1 reveal three important observations regarding the effect of inputs on execution time. Firstly, all three graphs show the sudden increase in execution time as the number of resource centers increases. As the number of resource centers reaches 8-9, there is an exponential increase in the time. In Fig 5.1 (a) and (b), the number of resources in each center is increased to 3 and 7 respectively. The trend in exponential increase persists over an increase in resources per center with a sudden surge observed around 7-9 centers. The second observation is that as the number of resources per center increases, the range of execution time increases significantly. The range of execution time when there are 3 resources per center is 0 -14 seconds in Fig. 5.1 (b) as compared to a range of 0 – 250 seconds when there are 7 resources per center in Fig. 5.1 (a). The third observation is that as the number of crises increases, the execution time for the same number of resource centers is higher. For example, in Fig. 5.1 (b), when the number of resource centers is 9, a 2-crisis scenario takes around 110 seconds and a 4-crisis scenario takes about 225 seconds. These observations typically aid a user in determining the range of an input set used to produce an optimal solution within an acceptable time frame.



(a)



(b)

Fig 5.1 Effect on Execution Time Due to Increase in Number of Resource Centers

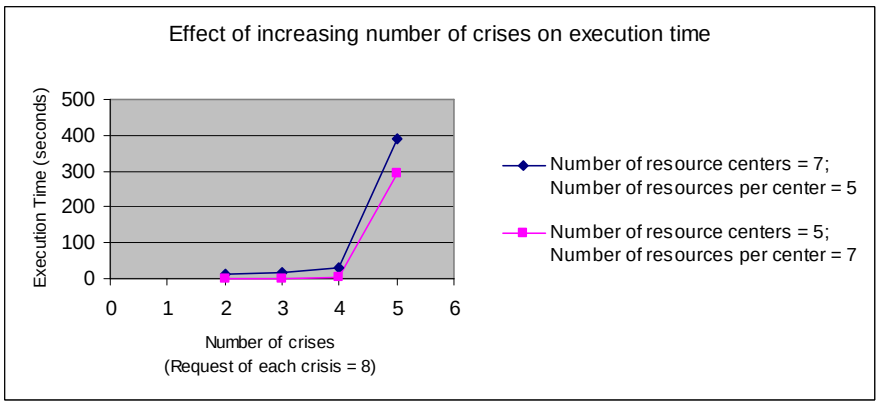


Fig 5.2 Effect of Increasing Number of Crises on Execution Time

Another factor which determines the execution time is the number of crisis locations. Fig 5.2 is a graph with a fixed number of total resources. Both the series show an increase in execution time as the number of crises increases. In the pink series, the number of resources centers is 5 with 7 resources per center and the blue series is vice versa. Although the total number of resources is constant, we observe that the series with 7 resource centers and 5 resources per center has a higher execution time with the same number of crises. For ease of plotting the graph, we have used the logarithm function to plot execution time.

For a fixed set of resource centers, capacities and request, as the number of crises increases, the size of a game increases thereby increasing the time taken to arrive at the Nash solution. Fig 5.3 shows this increase in the percentage of time taken to compute Nash Equilibrium as the dimension of the problem increases.

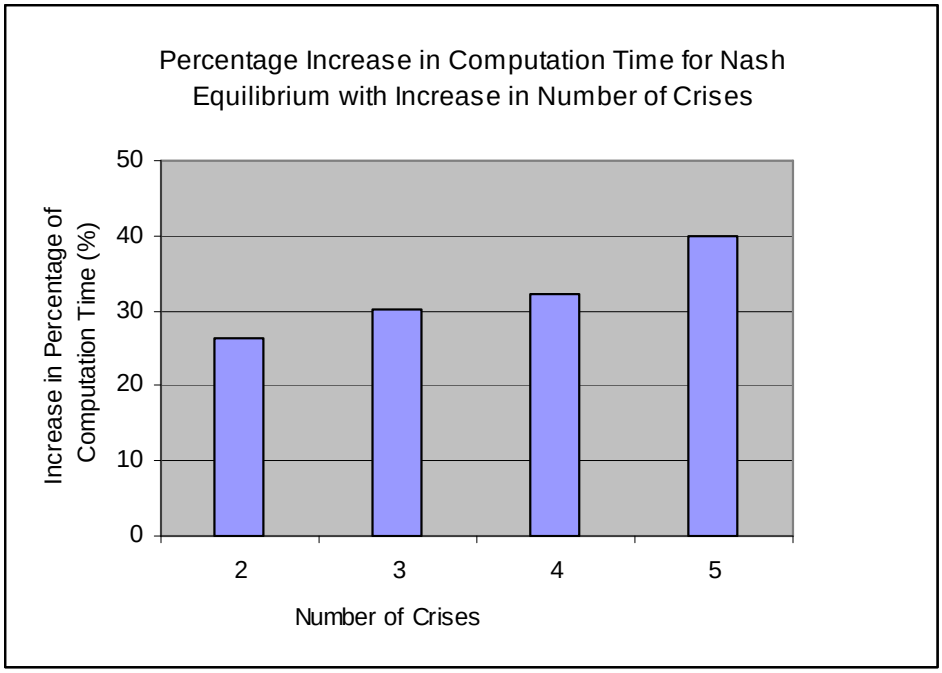


Fig 5.3 Percentage Increase in Computation time of Nash Equilibrium

5.3 Statistical Significance of Experimental Results for Execution Time

We perform regression analysis on our test data in order to determine the statistical significance of our results. We perform linear regression analysis by using the "least squares" method to fit a line through a set of observations. By doing so, we will be able to analyze how a single dependent variable is affected by the values of one or more independent variables.

Table 5.2 Regression Analysis Results

Independent Variables	Coefficient of independence	P- Value (Confidence)
Number of crises	3.59	2.64E-12
Number of resource centers	2.10	5.76E-13
Number of resources per center	0.61	0.25

In the linear regression model, the dependent variable is assumed to be a linear function of one or more independent variables plus an error introduced to account for all other factors. Consider a dependent variable *y* and an independent variable *x*, the coefficient of *y* on *x* is given by,

$$C = \frac{\sum xy}{x^2}$$

In our case, we examine execution time as the dependent variable and number of crises, number of resource centers and number of resources per center as the independent variables. We observe a significant impact of number of crises on the execution time. For every unit change in Log (Number of Crises), Log (Execution Time) increases by 3.592021. Similarly, for every unit change in Log (Number of resource centers) and Log (Number of resources per center), Log (Execution Time) increases by 2.10 and 0.61 respectively.

The p-value is the probability of finding a value as extreme or more extreme is a chance given that the null hypothesis is true. A null hypothesis basically assumes that none of the variables have any effect on the execution time. In our case, the p-values for number of crises, number of resource centers and number of resources per center are 2.64E-12, 5.76E-13 and 0.25.

In this chapter, we have examined the various aspects of our solution. We have quantified the fairness of our implementation and found fairness measures as high as 0.89. We derived significant inferences regarding the relationship between the various input parameters – crisis requests, resource availabilities and number of resource centers. Finally, we evaluated our test cases to understand the dependent and independent variables in the system and also verified the confidence of our test cases.

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

Crisis management has gained considerable importance over the last few years and the automation of allocation of emergency services is a logical step toward erasing human error. Each crisis in a multi-crisis scenario makes a request for a certain number of resources. A conflicting situation occurs when there is a shortage of resources or competition for resources from the same center. Although, each crisis has varying degrees of severity it is highly essential to cater to each crisis' request in the best possible manner. We have proposed an approach using game theory to allocate an optimal number of resources in a multi-crisis environment. Our method is a novel way of modeling a crisis scenario in a game theoretic framework and obtaining an allocation of resources that benefits all the crises in the game.

We have examined the effects of various input parameters like number of crises, resource capacities etc on the execution time. We have examined the effect of increasing crisis on the overall execution time of the system. Although, the implementation is affected significantly by the dimensions of the inputs, it has shown a degree of fairness of up to 0.89 in its results and can be used as a basis for modeling other resource allocation problems and obtaining feasible solutions. We have performed a linear regression analysis on our test cases and found the degrees of dependence between the variables and verified the confidence of our test cases.

6.2 Future Work

In the experiments that we performed, we restricted ourselves in the number of resource centers and the number of resources per center due to constraints on execution time. We need to investigate additional schemes to prune strategy spaces more effectively and improve the definition of a strategy in order to enhance the performance of the algorithm with increased dimensionality of the input set. The process of generation of strategies provides scope for parallelism which could improve execution time. Also, we need to explore additional factors that can be incorporated to enrich the payoff function like traffic delays, crisis growth probability, etc in order to obtain the best possible representation of a real life crisis environment and improve the quality of the solution.

REFERENCES

[1] T. Ibaraki and N. Katoh, Resource Allocation Problems. The MIT Press, 1988

[2] W. Karush, "A queuing model for an inventory problem", *J ORSA* 5 (1957), 693-703

[3] A. Charnes and W.W Cooper, "The theory of search Optimal distribution of effort", Manage Set 5 (1958), 44-49

[4] W. Karush, A general algorithm for the optimal distribution of effort, Manage Set 9 (1962), 0-72

[5] B.O. Koopman, "The optimum distribution of effort", *J ORSA* vol. 1 (1953) 52-63

[6] H. Luss and S.K. Gupta, "Allocation of effort resources among competitive activities", *Operations Research*, vol. 23 (1975), 360-366

[7] Z. Galil and N. Megiddo, "A Fast Selection Algorithm and the Problem of Optimal Distribution of Effort", *Journal of the ACM*, pages 58-64, 1979

[8] I. S. Gradshteyn and I. M. Ryzhik, "*Tables of Integrals, Series, and Products*", 6th ed. San Diego, CA: Academic Press, p. 1132, 2000

[9] N. Metropolis, A.W. Rosenbluth, M. N. Rosenbluth, A. H Teller and E. Teller, "Equations of State Calculations by Fast Computing Machines", *Journal of Chemical Physics*, vol. 21, 1087- 1092, 1958

[10] L. Yangsheng, S. Weiming and H. Ghenniwa, "A desired load distribution model for agent based distributed scheduling", *IEEE International Conference on Systems,Man and Cybernetics*, vol. 2, pp:1229 – 1234, Oct. 2003

[11] K. Iriuchijima, H. Sakamoto and M. Fujihara, "WIP allocation planning for semiconductor factories", *Proceedings of the 37th IEEE Conference on Decision and Control*, vol. 3, pp. 2716 – 2721, Dec. 1998

[12] D. L. Rhodes and W. Wolf, "Allocation and data arrival design of hard real-time systems", *Proceedings of the IEEE International Conference on Computer Design*, pp. 393– 399, 1997

[13] G. Yang and V. Kapila, "A dynamic-programming-styled algorithm for time-optimal multi-agent task assignment", *Proceedings of the 40th IEEE Conference on Decision and Control*,vol. 2 , pp. 1959 – 1964, 2001

[14] R. Bellman, "*Dynamic Programming*". Princeton, NJ, Princeton Univ. Press 1957

[15] "Dynamic programming", S. M. Lavalle, class notes for Department of Electrical and Computer Engineering, University of Illinois at Urbana Champaign

[16] X. Dahai, Y. Xiong and Q. Chunming, "Novel algorithms for shared segment protection", *IEEE Journal on Selected Areas in Communications*, vol. 21 , Issue: 8, pp.1320 – 1331, 2003

[17] F. Changqing and K. Wilken, "A faster optimal register allocators", *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 245 – 256, 2002

[18] R.N. Tomastik, P.B. Luh, L. Guandong, "Scheduling flexible manufacturing systems for apparel production", *IEEE Transactions on Robotics and Automation*, vol. 12, issue 5, pp. 789 – 799, Oct. 1996

[19] G. M. Chiu, C.S. Raghavendra, "A model for optimal resource allocation in distributed computing systems", *Proceedings of the 7th Annual Joint Conference of the IEEE Computer and Communcations Societies*, 1988 Pages:1032 – 1039

[20] D.Dutta, A. Goel and J. Heidermann, "Oblivious AQM and Nash Equilibria", ACM *Computer Communications Review*, 2002, vol. 32

[21] A.T. Rextin, Z. Irfan, Z.A. Uzmi, "Games networks play a game theoretic approach to networks", *Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks*, 2004, pp. 451 – 456

[22] E. Rasmusen, "*Games and Information*", Oxford,OX, UK,New York, N.Y.:B. Blackwell, 1989

[23] J. Nash, "Non-Cooperative Games", *The Annals of Mathematics*, 2nd Ser., vol. 54, No. 2. (Sep., 1951), pp. 286-295

[24] http://planetmath.org/encyclopedia/ExtensiveFormGame.html, 2004

[25] W. F. Lucas, "Some Recent Developments in n-Person Game Theory", *SIAM Review*, Vol. 13, No. 4, pp. 491-523

[26] M. Pincus, "A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems", *Operations. Research*, vol.18, 1225-1228, 1970

[27] S.Kirkpatrick,C.D. Gerlatt Jr. and M.P.Vecchi, "Optimization by Simulated Annealing", *Science*, vol. 220, 671-680, 1983

[28] "*Mathematical Optimization*", Computational Science Education Project Copyright (C) 1991, 1992, 1993, 1994, 1995 by the

[29] J.F. Puget, "Object oriented constraint programming for transportation problems", *IEEE Colloquium on Advanced Software Technologies for Scheduling*, 1993, pp. 411 – 413

[30] Q. Hao; B.H. Soong; E. Gunawan, J.T. Ong; C.B. Soh, L. Zheng, "A low-cost cellular mobile communication system: a hierarchical optimization network resource planning approach", *IEEE Journal on Selected Areas in Communications*, vol. 15 ,issue. 7, pp. 1315 – 1326, Sept. 1997

[31] R. Devadas, A.R. Newton, "Algorithms for hardware allocation in data path synthesis", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, issue. 7, pp. 768 – 781, July 1989

[32] J.E. Beck, D.P. Siewiorek, "Simulated annealing applied to multicomputer task allocation and processor specification", *8th IEEE Symposium on Parallel and Distributed Processing*, 1996, pp. 232 – 239

[33] H. Takahama, T. Nishi, M. Konishi, J. Imai, "A determination method of product allocation schedule for warehouse management ", *Proceedings of the 41st SICE Annual Conference*, vol. 2, 5-7 Aug. 2002, pp. 1004 - 1007

[34] N. Wattanapongsakorn, S.P. Levitan, "Reliability optimization models for embedded systems with multiple applications", *IEEE Transactions on Reliability*, vol. 53, issue. 3, Sept. 2004, pp. 406 – 416

[35] S. Kirkpatrick, "Optimization by Simulated Annealing - Quantitative Studies", *Journal of Statistical Physics*, vol. 34, 975-986, 1984

[36] J. H. Holland, "*Adaptation in Natural and Artificial Systems*", University of Michigan Press, Ann Arbor, MI, 1975

[37] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, "*Introduction to Algorithms"* (Second Edition), published by MIT Press and McGraw-Hill, 2004

[38] M.R. Sherif, I.W. Habib, M. Nagshineh, P. Kermani, "Adaptive allocation of resources and call admission control for wireless ATM using genetic algorithms", *IEEE Journal on Selected Areas in Communications*, vol. 18 , issue. 2, Feb. 2000, pp. 268 – 282

[39] S. Papavassiliou, A. Puliafito,O. Tomarchio, J. Ye, "Mobile agent-based approach for efficient network management and resource allocation: framework and applications", *IEEE Journal on Selected Areas in Communications*, vol. 20 , issue. 4, May 2002, pp. 858 – 872

[40] L. Wang; S. Wei, "Interconnect reuse based resource allocation with GA approach", *Proceedings of the 5th International Conference on ASIC*, 2003, vol. 1, issue. 21-24, Oct 2003, pp. 290 - 293

[41] S. Palaniappan, S. Zein-Sabatto, A. Sekmen, "Dynamic multiobjective optimization of war resource allocation using adaptive genetic algorithms", *Proceedings of IEEE*, SoutheastCon, 2001, vol. 30, pp. 160 – 165

[42] M. Asvial, B.G. Evans, R. Tafazolli, "Evolutionary genetic DCA for resource management in mobile satellite systems", *Electronics Letters*, vol. 38, issue. 20, 2002, pp. 1213 – 1214

[43] G. Attiya, Y. Hamam, "Two phase algorithm for load balancing in heterogeneous distributed systems", *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, 2004, pp. 434 – 439

[44] Task allocation algorithms for maximizing reliability of distributed computing systems Kartik, S.; Siva Ram Murthy, C.;Computers, IEEE Transactions on , 1997, vol. 46, issue. 6, pp. 719 – 724

[45] D.T. Peng, K.G. Shin, T.F. Abdelzaher, "Assignment and scheduling communicating periodic tasks in distributed real-time systems", *IEEE Transactions on Software Engineering*, vl. 23 ,iIssue. 12 , pp. 745 – 758

[46] L.G. Casado, I. Garcia, "Work load balance approaches for branch and bound algorithms on distributed systems", *Proceedings of the Seventh Euromicro Workshop on Parallel and Distributed Processing*, 1999, pp.155– 162

[47] Wikipedia, Encyclopedia; URL: http://en.wikipedia.org/wiki/Greedy_algorithm

[48] K.C. Almeroth, A. Dan, D. Sitaram, W. Tetzlaff, "Long term resource allocation in video delivery systems",. *Proceedings of the sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, 1997, vol. 3, pp. 1333 - 1340

[49] D. Kivanc, L. Guoqing, L. Hui, "Computationally efficient bandwidth allocation and power control for OFDMA", *IEEE Transactions on Wireless Communications*, 2003, vol. 2 , issue. 6, pp. 1150 – 1158

[50] V.M. Lo, "Heuristic algorithms for task assignment in distributed systems", *IEEE Transactions on Computers*, 1988, vol. 37, issue. 11, pp. 1384 – 1397

[51] S. Meltzin, Y. Zlotnikov, B.Z. Shklyar, "Optimal allocation of ATM networks resources by using a dynamic programming approach", , 1996., *Nineteenth Convention of Electrical and Electronics Engineers in Israel*, 1996, pp. 1 – 4

[52] L. Pronzato, L, "Optimal and asymptotically optimal decision rules for sequential screening and resource allocation", IEEE Transactions on Automatic Control, 2001, vol. 46, issue. 5, pp. 687 - 697

[53] P.B. Miao, X.Y. Chang, S.C. Castanon, D.A. Luh, "Stochastic task selection and renewable resource allocation", *IEEE Transactions on Automatic Control*, 1989, vol. 34, issue. 3, pp.335 – 339

[54] D.A. Castanon, J.M. Wohletz, "Model predictive control for dynamic unreliable resource allocation", *Proceedings of the 41st IEEE Conference on Decision and Control*, 2002, vol. 4 , pp. 3754 - 3759

[55] A.C.Fu, E. Modiano, J. N. Tsitsiklis "Optimal energy allocation and admission control for communications satellites", IEEE/ACM Transactions on Networking (TON), 2003, vol. 11 issue 3

[56] C.Y. Huang, J.H. Lo, S.Y. Kuo. M.R. Lyu, "Optimal allocation of testing resources for modular software systems." ,*Proceedings of the 13th International Symposium on Software Reliability Engineering*, 2002, pp. 129 – 138

[57] C.Y. Huang, J.H. Lo, S.Y. Kuo, M.R. Lyu, "Optimal allocation of testing-resource considering cost, reliability, and testing-effort", *Proceedings of the10th IEEE Pacific Rim International Sympsium on Dependable Computing*, 2004, vol. 3-5, March 2004, pp. 103 – 112

[58] H.C. Chao, R.C. Wang, J.L. Chen, "Fair resource sharing scheme for cellular data services with differentiated QoS", *IEEE Global Telecommunication Conference*, LOBECOM, 2000, vol. 1,pp. 266 - 270

[59] S. Gertphol, V.K. Prasanna, "MIP formulation for robust resource allocation in dynamic real- time systems", *Proceedings of the International Parallel and Distributed Processing Symposium*, 2003, pp8

[60] R. Rajkumar, C. Lee, J.P Lehoczky, D.P Siewiorek, "Practical solutions for QoS-based resource allocation problems", *Proceedings of the 19th IEEE Real-Time Systems Symposium*, 1998, pp.296 – 306

[61] A.E. El-Abd, "Modeling resources allocation and performance measures in distributed computer networks", *International Conference on Information Engineering [in conjunction with the] Proceedings of IEEE Singapore International Conference on Networks*, 1995, pp. 581 – 586

[62] X. Yizhi, X. Dahai, Q. Chunming, "Achieving fast and bandwidth-efficient shared-path protection", *Journal of Lightwave Technology*, vol. 21 , issue. 2 , 2003, pp. 365 – 371

[63] S. Gertphol, V.K. Prasanna, "Iterative integer programming formulation for robust resource allocation in dynamic real-time systems", *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, 2004, pp. 118 –125

[64] C. Reuter, M. Schwiegershausen, P. Pirsch, "Heterogeneous multiprocessor scheduling and allocation using evolutionary algorithms ", *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors*, 1997, pp. 294 – 303

[65] A.S. Al-Mahmeed, "Commercial applications of tabu search heuristics", *IEEE International Conference on Systems, Man and Cybernetics*, 1998, vol. 3, pp. 2391 - 2395

[66] S. Zhang, B. Ramamurthy, "Dynamic traffic grooming algorithms for reconfigurable SONET over WDM networks", *IEEE Global Telecommunications Conference*, 2002, vol. 3, pp.2716 - 2720

[67] A. Lim, B. Rodrigues, F. Xiao, Y. Z. Crane, "Scheduling using tabu search", *Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence*, 2002, pp. 146 – 153

[68] K. Nara, Y. Hayashi, "A solution algorithm based on multi-stage tabu search for nested combinatorial optimization problem",*IEEE International Conference on Systems, Man and Cybernetics*, 1999, vol. 3, pp. 551 – 556

[69] T. Hansen, "On the Approximation of Nash Equilibrium Points in an N-Person Noncooperative Game", *SIAM Journal on Applied Mathematics*, vol. 26, pp. 622-637

[70] H. E. Scarf, T. Hansen, "*The computation of economic equilibria*", Cowles Foundation Monograph, Yale University Press

[71] http://www.appsci.com/cms/cms.htm

[72] http://www.infospheres.caltech.edu/crisis_web/overview.html

[73] T. Alpcan, T.A. Basar, "A game-theoretic framework for congestion control in general topology networks", *Proceedings of the 41st IEEE Conference on Decision and Control*, 2002, vol. 2, pp. 1218 - 1224

[74] Gate sizing and buffer insertion using economic models for power optimization Murugavel, A.K.; Ranganathan, N.; *Proceedings of the 17th International Conference on VLSI Design*, 2004 Pages:195 – 200

[75] D. Galati, Y. Liu, M.A. Simaan, "A fast algorithm for unit level team resource allocation in a game environment", *Proceedings of the 42nd IEEE Conference on Decision and Control*, 2003, vol. 3, pp. 2872 - 2877

[76] L. Libman, A. Orda, "Atomic resource sharing in noncooperative networks, *Proceedings of 16th Annual Joint Conference of the IEEE Computer and Communications Societies*, 1997, vol. 3, pp. 1006 - 1013

[77] D.E. Pendarakis, D.E.;A.A. Lazar, A Orda, "Virtual path bandwidth allocation in multi-user networks", *Proceedings of the 14th Annual Joint Conference of the IEEE Computer and Communications Societies*, 1995, pp. 312 - 320

[78] N. Feng, S.C. Mau, N.B. Mandayam, "Pricing and power control for joint network-centric and user-centric radio resource management", *IEEE Transactions on Communications*, 2004, vol. 52, issue. 9, pp. 1547 – 1557

[79] R. Elliott, "A measure of fairness of service for scheduling algorithms in multi-user systems", *IEEE Canadian Conference on Electrical and Computer Engineering*, 2002, vol. 3, pp. 1583 - 1588