A CLOSED FORM OPTIMIZATION MODEL FOR THE CONFLICT NEUTRALIZATION PROBLEM

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Yan Wang

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

November 2010

Fargo, North Dakota

North Dakota State University
Graduate School

Title

A CLOSED FORM OPTIMIZATION MODEL FOR THE

CONFLICT NEUTRALIZATION PROBLEM

By

YAN WANG

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

ABSTRACT

Wang, Yan, M.S., Department of Computer Science, College of Science and Mathematics, North Dakota State University, November 2010. A Closed Form Optimization Model for The Conflict Neutralization Problem. Major Professor: Dr. Kendall Nygard.

In this study, we proposed a novel closed form optimization model for the Conflict Neutralization Problem (CNP) and implemented an efficient algorithm for solving the problem. A novel tableau representation of the CNP model was presented and described in detail. We implemented a special structured branch and bound algorithm to solve the problem. Key components of the implementation were described. To test the computation performance of our algorithm, we designed and conducted three sets of experiments. The experiment results were reported and analyzed in this report. The test results showed the efficiency of the algorithm for solving the Conflict Neutralization Problem.

ACKNOWLEDGMENTS

I would like to express my gratitude to Dr. Kendall Nygard for his guidance and advice during my study at North Dakota State University. I would like to thank him for providing me with valuable advice and extensive knowledge. I would also like to gratefully acknowledge my supervisory committee, Dr. William Perrizo, Dr. Jun Kong and Dr. Wei Lin, for their valuable advice and comments.

DEDICATION

To my parents for their unlimited love and support during my master's study, for the tough time they spent with me in a foreign land. The past three years have been the most challenging years of my academic and personal life. Without my parents, I would not be where I am.

The special thanks belong to my husband, Tingda, and my son, Michael, whose love and encouragement made this thesis possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

In this study, a novel closed form approach for solving the Conflict Neutralization Problem (CNP) is proposed. The CNP was first modeled as Generalized Assignment Problem (GAP) through a generalizing of tasks and agents in GAP, for which many efficient algorithms exist. Then a Branch and Bound algorithm was modified and implemented for solving the CNP. To further improve the performance of the algorithm, we exploited special structures in the implementation of it. In this chapter, we will introduce the CNP, the mathematical definition of the CNP, the GAP used to solve the problem, the Branch and Bound algorithm and some ideas of the experiment results of our solver.

1.1. Problem Statement

We assume that within a geographical area, there are conflict areas emerging and dynamically increasing or decreasing in severity. The conflict areas could be of different types, and could be such things as fires, terrorist cells, or military enemy units. There are sensor platforms and other data sources available to provide information about the severity, locations and types of conflict areas. The information provided by the heterogeneous sources is dynamically changing and is typically partial and/or imprecise. Response units that are equipped to neutralize conflict areas are available, and vary in their mix of resources and in the types of conflicts that they can address. The resources of the response units are reconfigurable and can be redistributed among themselves. The response units may also configure themselves into teams to neutralize the conflict areas. They stationed initially at different locations and may differ in mobility factors, such as speed and endurance. The problem objective is to allocate and dispatch the resources and the tasks to response units adaptively over time. We coin the term Conflict Neutralization Problem or CNP to refer to the problem of interest.

In the CNP, it is desirable that more than one response unit be assigned to a single task to deal with situations, where differing types of resources are useful in neutralizing a conflict area. Here we constraint each response unit to engage with only one conflict area. The cost for teams that have different team sizes to neutralize a conflict area is provided by each individual team by running subservient models aimed at optimizing the coordination of their team missions.

The Conflict Neutralization Problem can be mathematically defined as below:

Minimize:

$$z = \min \sum_{k=1}^{l} \sum_{i=1}^{m} v_{i,k} y_{i,k} \tag{1}$$

Subject to :

$$\sum_{i=1}^{m} \sum_{k=1}^{l} x_{i,j,k} \leq 1 \qquad j = 1, ..., n; \tag{2}$$

$$\sum_{j=1}^{n} x_{i,j,k} - k * y_{i,k} = 0 \qquad i = 1, ..., m, k = 1, ..., l; \tag{3}$$

$$\sum_{k=1}^{l} y_{i,k} \leq 1 \qquad i = 1, ..., m; \tag{4}$$

$$y_{i,k} = 0, 1 \qquad i = 1, ..., m, k = 1, ..., l; \tag{5}$$

$$x_{i,j,k} = 0, 1 \qquad i = 1, ..., m, j = 1, ..., n, k = 1, ..., l; \tag{6}$$

Where:

i is conflict areas {1, 2, ..., m}.

j is response unit {1, 2, ..., n}.

k is team size {1, 2, ..., l}.

$v_{i,k}$ is the objective function coefficient of assign a team size k to a conflict area.

Variable $y_{i,k}$ represents the assignment of a team size k to a conflict area i.

Variable $x_{i,j,k}$ represents the assignment of a response unit j to neutralize a conflict area i in a team with team size k.

Constraint (2) means that one response unit can be engaged with only one or no conflict area. For the engaged conflict area, a response unit can only be in a team with team size k. For example, response unit j can be involved in a team of size 1, 2, ... l, and the team can only be assigned to neutralize only one conflict area or the response unit can be dismissed and not involved in neutralization of any conflict area.

Constraint (3) means that for one conflict area i, if one team size k is assigned to neutralize it, then there must be k response units assigned to neutralize it.

Constraint (4) means that for any conflict area, only one team size can be chosen.

Constraint (5) means the decision variable $y_{i,k}$ is binary.

Constraint (6) means the decision variable $x_{i,j,k}$ is binary.

Solving the CNP problem is non-trivial. Because of the constraints (2), (3) and (6), the formulation above cannot be solved with standard linear programming techniques. In this study, we will show that the problem can be formulated and solved as GAP. Since there exists efficient algorithms for the GAP, we can solve the problem efficiently with some special data structures.

1.2. GAP Used to Address CNP

To solve the CNP, we first model the problem as a special Generalized Assignment Problem (GAP). In GAP, there are a number of agents and tasks. An agent can be assigned to perform any task in the task set. An assignment of an agent to a task will incur some costs. The profit of an assignment varies depending on the agent-task pair. Each agent has a capacity limit. The sum of the costs of assign a task to it cannot exceed this limit. The solution to a GAP problem is the one that all agents do not exceed their budget and total profit of the assignment is maximized. For the problem presented in this paper, it is necessary to solve the minimum GAP problem shown below:

Minimize:

$$z = \min \sum_{i=1}^{m} \sum_{j=1}^{n} v_{i,j} x_{i,j} \tag{7}$$

Subject to :

$$\sum_{i=1}^{m} x_{i,j} = 1 \qquad j = 1, ..., n; \tag{8}$$

$$\sum_{j=1}^{n} r_{i,j} x_{i,j} \leq b_i \qquad i = 1, ..., m; \tag{9}$$

$$x_{i,j} = 0 \; or \; 1 \qquad i = 1, ..., m, j = 1, ..., n; \tag{10}$$

In the formulation, j is the set of tasks, i is the set of agents, $r(i, j)$ is a measure of the amount of resource that agent i spends in carrying out task j. $b(i, j)$ is the total resource that is available to agent i and $v(i, j)$ is the the cost to assign a agent i to carry out task j. The decision variables $x(i, j)$ are binary and model the assignment or non-assignment of tasks to agents. The objective function accumulates and provides the total cost of the tasks being assigned to agents. The constraints set (8) requires that each task is done by one agent and the constraint set (10) enforces that each agent's resource capacity is not exceeded.

The formulation above is generic. In the CNP it is desirable that more than one response unit (RU) be allowed assignment to a single task, to deal with situations where different types of resources are useful in neutralizing a conflict area (CA) and the resources are only available at different response units. For example, there could be situations in which specific teams comprised of RUs working together to increase the probability of neutralizing the conflict area. The cost is expected to be nonlinear in the number of RUs. To function as teams, the RUs must be in close communication with each other, and negotiate to agree upon a feasible estimated time of arrival (ETA) based on their specific role in neutralizing the conflict area.

In this work we proposed a novel modeling techniques to fit the CNP to a GAP formulation. We will present the detail of the model in Chapter 3.

1.3. Method Used to Solve the GAP

The GAP is a NP-hard program. But due to the difficulty in solving "hard" GAPs, in the case of optimizing methods, computational results are limited to 500 to 1,000 binary variables in the beginning. A considerable body of literature exists on the search for effective enumeration algorithms to solve GAP problems. Both exact and heuristic algorithms have been suggested for solving the GAP problem. Various mathematical programming techniques have been tried on the GAP problem including branch and bounds techniques, relaxation, dual approaches, meta-heuristics, set partitioning, etc. A large number of special GAP problems have more efficient algorithms designed to take advantage of their special structures.

The branch and bound is an exact method for finding an optimal solution to an NP-hard problem. It is an enumerative technique that can be applied to a wide class of combinatorial optimization problems. The branch and bound algorithm makes the search much more efficient by using bounds on the objective function to prune large parts of the search tree. For example, a branch and bound algorithm for integer optimization problem begins by solving the continuous relaxation of the original problem. If a 0/1 variable is fractional, the algorithm constructs two new subproblems. In one of the subproblem the variable is set to be zero and another the variable is one. The algorithm proceeds in this fashion by solving subproblems and creating new subproblems until an integer solution has been found and each remaining subproblem has a higher lower bound than the integer solution.

In this study, we present a special structured branch and bound algorithm for solving the CNP. The elements of the algorithm are described in Chapter 4.1 to 4.3. In Chapter 5, we will describe the implementation. Computation experiments and results are reported in Chapter 6.

1.4. Analysis of Experiments Results

Due to the special structured model of the CNP, existing computer programs are not able to process the model and few computer programs are available to offer modeling flexibility for the problem. Also, existing commercial mixed integer linear programming products are too general to be efficient on large sized problems like the CNP. In Ross and Soland's study, they programmed and tested some special structured model with their algorithms. The calculation time is not as small as other's work. For solving the CNP, we implemented the branch and bound algorithm of Ross. In this study, we implemented an efficient branch and bound algorithm with C++ and solved a variety of test problems with randomly generated variables. The computational results for problems of different size are reported in Chapter 6. The standard deviation, median, minimum and maximum calculation times are reported, as well as the median values of the number of feasible solutions and nodes.

To demonstrate the efficiency of our algorithm, we solved sets of randomly generated CNPs. The computational results clearly show the efficiency of our algorithm for the CNP. As indicated in the computational results, the calculation times of our algorithm are relatively stable for each problem size. As problem size increases, the calculation time increases but at a relatively slow rate when the number of variables is less than certain limit.

We also noticed that the number of dismissible response units affects the scalability of the algorithm. If the dismissible response unit is low, most of the response unit must be involved in neutralizing the conflict area. Compared with problem with a higher number of dismissible response units, low dismissible response units impose more restriction on the problem and result in more complex computation.

CHAPTER 2. LITERATURE REVIEW

In this chapter we will give a brief review of the relevant literature on Conflict Neutralization Problem, Branch and Bound techniques, etc.

As John Arquilla and David Ronfeldt stated, the information revolution is altering the nature of low intensity conflict, crime, and activism waged by actors like terrorists, fire apparatus, etc. As the information revolution deepens, the conduct and outcome of conflicts increasingly depend on information and communications. The actors are organizing into sprawling, loose, leaderless networks [2]. From the 1990s, officials and analysts in U.S. and European government, military, and police circles began to show an interest in the concept. However, they found it difficult to deal with terrorists, criminals, and fanatics associated with militias and extremist single-issue movements, largely because these antagonists were organizing into sprawling, loose, leaderless networks, overcoming their former isolated postures as stand-alone groups headed by "great" person.

The CNP is a special case of pairing problem that shares the same feature. In the CNP, the information provided by the heterogeneous sources is dynamically changing and is typically partial or imprecise. The decision making and operations are less centralized, therefore allowing for local initiative and autonomy. Based on the communication and computation capability of actors, each team can run subservient models aimed at optimizing the coordination of their team missions. Typically in the CNP, the cost for teams with different team size to neutralize a conflict area is provided.

Since mid-fifties, many research articles on pairing problems have been published. Some of these problems are easy to solve, whereas others are extremely difficult like the Generalized Assignment problem. The Generalized Assignment Problem is used in many applications such as assigning jobs to computers in a computer network

[3], vehicle routing [7] and plant location [11]. Numerous studies have been done on approaches for solving the GAP. In 1974 Geoffrion coined the term Lagrangian Relaxation in an early exposition of the concept and applied the Lagrangean Relaxation in obtaining lower bounds when solving a integer programming problem [8]. After that the Lagrangean Relaxation was applied to many discrete optimization problems including facility location and capacity planning, scheduling, vehicle routing and network flow, set covering, generalized assignment, and other variations on the knapsack problem. Beyond these traditional resource allocation problem, the techniques was also applied to problems like biology, electric power generation, the chemical and petroleum industries.

In 1975, Ross and Soland [11] proposed an efficient branch and bound algorithm which applies an elaborate technique for bounding calculation by solving a series of binary knapsack problems. In their algorithm, the bounding process starts from solving an initial simple and easy to calculate relaxation of the original problem. An initial bound is obtained by solving first relaxation problem. Then a set of PKi problem was created and solved for each agent i that the capacity constraints is violated. Each PKi problem is a binary knapsack problem. The solution to each PKi designates those tasks that must be reassigned from agent i to other agents in order to satisfy the resource restriction on agent i. The solution was used to further refine the lower bound by the sum of the values of the objective functions obtained.

Another benefit of solving the PKi problems is that by reassigning some of the variables $x_{i,j}$, a feasible solution to original problem P may be constructed. This new solution will be used for later pruning process. In their report, Ross and Soland also showed that their approach for bounding can be viewed as a specific application of Lagrangean Relaxation. The method for branching of their approach is based on the 0-1 dichotomy of variable values. The chosen variable is also based on the solution of previous PKi problem. The variable chosen represents a job that is best kept with agent i considering both the penalty for switching the job and the resources available to the agent.

Ross and Soland implemented their algorithm and solved a variety of test problems. They applied a LIFO (Last In First Out) treatment to the candidate problems for fast update of candidate list and small storage space. Based on the computation results, the algorithm could solve problems with up to 4000 variables and the computation time was relatively stable for each problem size in comparison to other algorithms tested.

Different metaheuristic approaches also have been proposed by other authors [1][4][6]. A survey of exact and heuristic algorithms to solve the GAP can be found in Cattrysse and Wassenhove's report [5].

A variety of extensions of the GAP have been proposed in the past in order to more closely describe various real-world applications. Srinivasan and Thompson [13] mentioned an extension of the model where the capacity of the agents can be increased at a certain cost, which is linear in the amount of capacity added. Neebe and Rao proposed a fixed-charge version of the GAP with agent-independent requirements where each agent processing at least one task incurs a fixed cost [10]. In 1989, Mazzola states that nonlinear capacity interaction can be found in hierarchical production planning GAP, where product families must be assigned to production facilities and a changeover is translated into nonlinear capacity interaction between the product families assigned to the same facility [9].

In 1977, Ross showed that many location problems can be formulated as special structured GAP and solved efficiently with existing algorithms [12]. He first introduced additional "tasks" and "agents" in the tableau representation of original locations problems as showed in Figure 1. This includes a large amount of new cells into the tableau representation. Ross and Soland elaborately defined the value of the $r_{i,j}$, a_i and b_i in the tableau as,

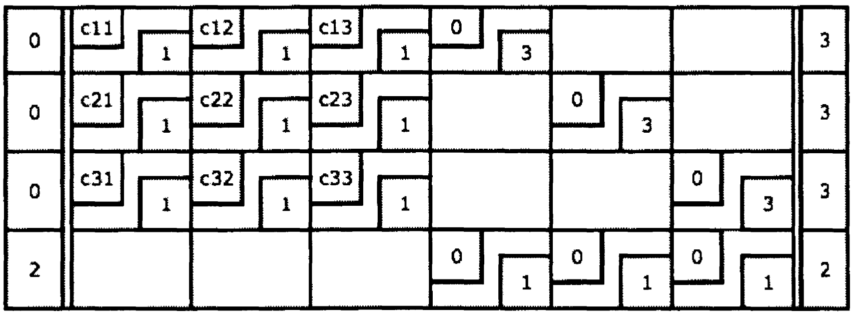
$$a_i = 0, \qquad b_i = N, \qquad i = 1, ..., N. \tag{11}$$

$$a_N + 1 = b_N + 1 = P, \tag{12}$$

The special values of these coefficients enables that the GAP constraints (13) and (14) can guarantee that each specific requirement of the original location problem are satisfied. For example, when constraint (13) and (14) applied, the last row of tableau in Figure 1 insures that exactly two demand centers are designated as supply centers.

$$\sum_{i=1}^{m} x_{i,j} = 1 \qquad j = 1, ..., n; \tag{13}$$

$$x_{i,j} = 0 \; or \; 1 \qquad i = 1, ..., m, j = 1, ..., n; \tag{14}$$



0	c11 1	c12 1	c13 1	0 3			3
0	c21 1	c22 1	c23 1		0 3		3
0	c31 1	c32 1	c33 1			0 3	3
2				0 1	0 1	0 1	2

Figure 1. GAP tableau representation of a p-median problem

In their study, they considered both public and private section models of facility location problems like p-median problem, capacity constrained p-median, capacitated warehouse location problems, etc. They implemented an existing algorithm to solve a set of example GAPs. As they stated, the computation time is not as small as some others reported in recent research.

In this study, we reviewed research works on optimization problems like location problems, Netwar, fire fighting, etc. A large number of special GAP problems have more efficient algorithms designed to take advantage of their special structure. Based on the nature of the CNP, we proposed a novel tableau representation of the Conflict Neutralization Problem, which was modeled as a GAP by generalizing of tasks and agents in GAP. To solve the CNP, we modified and implemented the branch and bound approach proposed by Ross. We further improved performance of the algorithm by applying special structures to exploit the special feature of the CNP model.

CHAPTER 3. MODELING CNP AS GAP

As stated in Chapter 1, the CNP problem cannot be solved with standard linear programming techniques. In this study, we showed that the problem can be formulated and solved as Generalized Assignment Problem (GAP), for which there exists efficient algorithms for the GAP. In this chapter, we will present detailed explanation of the CNP model.

3.1. Tableau Representation of CNP

Figure 2 shows how GAP constructs can support the global assigning of multiple response units to teams that work cooperatively in such a way that the total cost for neutralizing conflict areas is minimized.

		RU1	RU2	RU3	RU4	RU5	CA1 Single RU	CA1 Double RU	CA1 Triple RU	CA2 Single RU	CA2 Double RU	CA2 Triple RU	CA3 Single RU	CA3 Double RU	CA3 Triple RU	CA1 Neutralize	CA2 Neutralize	CA3 Neutralize	Upper Limit
0	CA1 Single RU	0 1	0 1	0 1	0 1	0 1	0 1												1
1	CA1 Double RU	0 1/2	0 1/2	0 1/2	0 1/2	0 1/2		0 1											1
2	CA1 Triple RU	0 1/3	0 1/3	0 1/3	0 1/3	0 1/3			0 1										1
3	CA2 Single RU	0 1	0 1	0 1	0 1	0 1				0 1									1
4	CA2 Double RU	0 1/2	0 1/2	0 1/2	0 1/2	0 1/2					0 1								1
5	CA2 Triple RU	0 1/3	0 1/3	0 1/3	0 1/3	0 1/3						0 1							1
6	CA3 Single RU	0 1	0 1	0 1	0 1	0 1							0 1						1
7	CA3 Double RU	0 1/2	0 1/2	0 1/2	0 1/2	0 1/2								0 1					1
8	CA3 Triple RU	0 1/3	0 1/3	0 1/3	0 1/3	0 1/3									0 1				1
9	CA1 #RU						V11 1	V12 1	V13 1							0 1			1
10	CA2 #RU									V21 1	V22 1	V23 1					0 1		1
11	CA3 #RU												V31 1	V32 1	V33 1			0 1	1
12	CA Dispatch															0 1	0 1	0 1	3
13	RU Dismiss	0 1	0 1	0 1	0 1	0 1													5

Figure 2. Modeling CNP as GAP

In the figure, columns correspond to generalized tasks. Rows are generalized agents. Cells correspond to decision variables $x_{i,j}$. Within a cell (i, j) of the table, the objective function coefficient $v(i, j)$ is shown as the top entry and the resource consumed by the assignment $r(i, j)$ is the bottom entry. Constraint (8) stipulated that for any feasible solution, exactly one of the cells in each column is chosen (multiple choice constraint). Across each row, the chosen cells must have a total resource consumption that does not exceed the upper limit (capacity), shown on the end of the row. When the upper limit is set to 1, the model solution will assign at most one task to the corresponding agent, i.e., a binary assignment structure. Empty cells in the tableau represent inadmissible assignments. These could be handled computationally through large penalty costs, but we prefer to handle them by employing specialized data structures in the solution algorithm to bypass these assignments.

3.2. Model Description

The CNP model discussed above supports scenarios that single tasks can be cooperatively carried out by multiple RUs. The value of engaging a CA is likely to be nonlinear in the number of RUs. The GAP framework supports extended modeling constructs for this type of scenario by generalizing the meaning of jobs and agents.

In the example of the tableau, there are five available RUs and three CAs. Each task can be assigned 0, 1, 2, or 3 RUs, i.e., the team size to neutralize the CA can be 0, 1, 2, 3. If the team size is more than one, the team members are granted the autonomy to coordinate their mission using models at the team level. For example, if different resources are required to neutralize an conflict area, or efficiency will be gained in scale if large numbers of response units are assigned to neutralize a conflict area.

The first five columns of the table correspond to generalized GAP tasks which model the response units. Based on GAP Constraint (8), exactly one cell is selected in each column, so each response unit is restricted to choose only one from the ten cells that are available.

The agent choices in the first three rows enforce that if conflict area CA1 is assigned, there will be one, two, or three response units. Upper limits for each of the first nine rows is set to 1. Because the variables are binary and the resource consumed are inversely proportional to the number of RUs in the team, each of these rows enforces a condition that either one of the specified number of RU choices is assigned to the CA, or none at all. The bottom row represents the option of the RU having no assignment.

Cells on the diagonal structure correspond to agent/task pairs for each possible team size for each CA. This mechanism allows for a given team size to not be selected. This induces a ripple effect in the model. More specifically, a non-selection forces a selection in the cell in the column corresponding to the task/number of RU agents enumerated in rows 9, 10, and 11. These cells are where the team reward values $v(i,j)$ are provided for the model. This structure models nonlinear returns to scale, because individual reward value entries for 1, 2, or 3 RUs are provided for each CA. The actual $v(i,j)$ values would be provided by the individual teams, through running subservient models aimed at optimizing the coordination of their team missions. Rows 9 - 11 have an upper limit of one, essentially making these multiple choice constraints. This enforces the assigning of exactly 0, 1, 2, or 3 RUs to each task. Finally, row 12 models selection or non-selection of each CA. The upper limit of 3 provides for selecting at most all three tasks, but also makes it possible to assign less than all three, which may be optimal for some scenarios, depending on the severity of the CA.

The model supports fundamental differences in the RUs through reward coefficients for the various available combinations. Thus, the model has remarkable flexibility for heterogeneous RUs with different resources or other differences. the potential drawback of the approach is computational solution time. However, to improve solution time, the enumeration solution scheme can take advantage of the considerable special structure.

CHAPTER 4. A BRANCH AND BOUND ALGORITHM FOR CNP

To solve the CNP, we modified and implemented the branch and bound approach proposed by Ross [11]. The algorithm solves the generalized assignment problem by solving a series of binary knapsack problems. Previous research work and our experiment result demonstrated the efficiency of the approach. In this chapter we will present the branch and bound procedure in detail.

4.1. Brief Introduction to Branch and Bound

Solving NP-hard discrete optimization problems to optimality requires very efficient algorithms. Branch and bound is by far the most widely used tool for solving large scale NP-hard combinatorial optimization problems. The branch and bound algorithm searches the complete solutions space for a given problem for the best solution. As the solution process, the status of the solution with respect to the search of the solution space is described by a pool of yet unexplored subset of this space and the best solution found so far.

The branch and bound procedure consists of three important components. The first one is a splitting procedure that split a set S of candidates into two or more smaller sets. The union of these sets covers original solution set S.

Another component is the bounding procedure that computes upper or lower bounds for the minimum value of within a given subset S. If the lower bound for a tree node is greater than the upper bound for some other node or the upper bound for a tree node is less than the lower bound, then that tree node can be safely discarded from the search. This component is called pruning.

The efficiency of a branch and bound algorithm depends heavily on the effectiveness of the branching and bounding approach. Poor design may result in repeated branching without any pruning.

In the branch and bound algorithm used in this study the bound is calculated in part by solving a set of binary knapsack problems, which require only minimal effort. The algorithm exploits the structure of the CNP problem and yields an integer solution that tends to be feasible. With a LIFO implementation of candidate problems, the algorithm demonstrated great efficiency.

Minimize:

(P)

$$z = \min \sum_{i=1}^{m} \sum_{j=1}^{n} v_{i,j} x_{i,j} \tag{15}$$

Subject to :

$$\sum_{i=1}^{m} x_{i,j} = 1 \qquad j = 1, ..., n; \tag{16}$$

$$\sum_{j=1}^{n} r_{i,j} x_{i,j} \leq b_i \qquad i = 1, ..., m; \tag{17}$$

$$x_{i,j} = 0 \ or \ 1 \qquad i = 1, ..., m, j = 1, ..., n; \tag{18}$$

4.2. Bounding Algorithm

The first step of the bounding procedure of the algorithm is the initial relaxation of problem (P). The relaxation is a natural one in which tasks are assigned to the least costly agent without regard to the limitations on multiple assignments. This relaxation is

Minimize:

(PR)

$$z = \min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{i,j} x_{i,j} \tag{19}$$

Subject to :

$$\sum_{i=1}^{m} x_{i,j} = 1 \qquad j = 1, ..., n; \tag{20}$$

$$x_{i,j} = 0 \; or \; 1 \qquad i = 1, ..., m, j = 1, ..., n; \tag{21}$$

The solution to (PR) is found by determining an index $i_j \in I$ for all $j \in J$, such that $c_{i_j,j} = min_{i \in I}\{c_{i,j}\}$ and setting $x_{i_j,j}$ and $x_{i,j} = 0$ for all $i \in I, i \neq i_j$.

The lower bound yields this way is:

$$Z = \sum_{j=1}^{n} c_{i_j,j} \tag{22}$$

Each of the constraints (18) in (P) is then checked for feasibility using the solution to (PR). Most of the time, some of these constraints will be violated. A set of (PK$_i$) problem are then solved to refine the lower bound.

Let

$$J_i \equiv \{j : x_{i,j} = 1, j = 1, ..., n\}$$

Let

$$I' \equiv \{i : \sum_{j=1}^{J_i} r_{i,j}x_{i,j} < b_i, i = 1, ..., m\}$$

The PK$_i$ problem is defined as below:

Minimize:

(PK$_i$)

$$Z_i = \sum_{j \in J_i} p_j y_{i,j} \tag{23}$$

subject to

$$\sum_{j \in J_i} r_{i,j} y_{i,j} \geq d_i \tag{24}$$

$$y_{i,j} = 0 \ or \ 1 \tag{25}$$

where

$$d_i = \sum_{j \in J_i} r_{i,j} x_{i,j} - b_i$$

$$p_j = \min_{k \in I - \{i_j\}} \{c_{k,j} - c_{i_j,j}\}$$

The optimal solution to (PK$_i$) indicates those reassignments that lead to a minimal increase to the value of Z. The revised lower bound for (P) is:

$$LB = Z + \sum_{i \in I'} z_i^*.$$

An additional refinement can be made in the procedure for solving (PK$_i$) as the branching progresses, particularly as the algorithm proceeds down the "one-branch".

$$r_{i,j} > b_i - \sum_{j \in F_i} r_{i,j} x_{i,j}$$

where F_i designates those tasks assigned to agent i in the current candidate problem.

This may increase the bound and penalties p_j in the objective function of the (PK$_i$).

4.3. Branching Algorithm

The variable x_{i^*,j^*} that chosen to separate on is decided based on;

1. The variable should be one from the solution set for (PK$_i$) with $y_{i,j}^*= 0$;

2. $t_{i^*,j^*} = \max\{p_j/r_{i,j}/(b_i - \sum_{j \in F_i} r_{i,j}x_{i,j})\}$,

where F_i denotes the set of tasks assigned to agent i.

In this way, the solution set for (P) is separated into two mutually exclusive and collectively exhaustive subsets based on the 0-1 dichotomy of variable values.

The branching approach above creates two new candidate problems whose solution sets differ only in the value assigned to a particular variable. The candidate problem in which the separation variable is fixed to one is the problem examined next.

4.4. Pruning Criterion

The solutions of the binary knapsack problems in (PK_i) indicate some reassignments of tasks to other agents that may lead to a feasible solution. Those feasible solutions are used in our algorithm for search space pruning. A variable that records the minimum upper bound seen among all subtrees examined so far was maintained globally. If the lower bound for some set of candidates is greater than that global upper bound, then that candidates can be safely discarded from the search.

In this study, we implemented the branch and bound algorithm presented above with special data structures, the experiment results shows good efficiency and scalability of the algorithm.

CHAPTER 5. ALGORITHM IMPLEMENTATION

The branch and bound algorithm was implemented with C++ under Redhat 5.0. In the implementation, we apply a LIL treatment on the bounding process of the algorithm. The LIL treatment greatly reduced the processing time by avoiding process of inadmissible cells in the model. Algorithm 1 shows the general branch and bound procedure.

```
Input: nCAs(Number of Conflict Areas), nRUs(Number of response unit),
       nSizes(Number of team size)
Output: Optimum Solution
$CreateModel(nCAs, nRUs, nSizes)$;
$CnpSolver.initialize()$;
while $pStackHead$ and $nSerialNo \leq MAX\_NODES$ do
    $pCur = pop(pStackHead)$;
    $prSolve()$;
    if $feasible(\ )$ then
        update optinum ...;
        $fanthom()$;
    else
        $pkSolver()$;
        $reassign()$;
        if $feasible()$ then
            update optinum ...;
            $fanthom()$;
        else
            Create NewNode0;
            Create NewNode1;
            push stack NewNode0;
            push stack NewNode1;
        end
    end
end
```

Algorithm 1: Branch and Bound for the CNP

5.1. Definitions and Data Structures

As the tableau representation of the CNP model exists large number of inadmissible cells, we exploit this special structure in implementing the branch and bound algorithm. List of lists are created for storing of the objective coefficient and resource consumption, LIL stores one list per row, where each entry stores a column index and value. The LIL treatment of the model greatly reduce the computation in solving the PR problems and the PK$_i$ problems.

```
/* List of lists node */
typedef struct LILNode{
        int nValue;                 /*Value of the cell*/
        int nIndex;                 /*Column or row number of the cell, */
        LILNode * pNext;            /*Pointer to next cell*/
};
```

the branch and bound tree consist of nodes defined below. Each node points to its parent through the pParent pointer, except for the root. When solving the (PR) and (PK$_i$) problem, by following the path from a particular node to the root, the algorithm can determine all the variables that are fixed to either zero or one. The pPrev pointer link node to its previous node in the LIFO stack. The branch and bound algorithm will terminate when the stack is empty.

```
/* Branch and bound node structure */
typedef struct Node {
        int nNode;                  /*Node no*/
        int nBound;                 /*Lower bound of the node's LP relaxation*/
        int nAgentFixed;            /*Agents fixed*/
```

```
    int nTaskFixed;              /*Tasks fixed*/
    int nFixedTo;                /*Variable fixed to 1 or 0*/
    struct Node *pParent;        /*Pointer to parent Node*/
    struct Node *pPrev;          /*Pointer to previous node
                                 in the branch and bound stack*/
} Node;
```

Global definations:

```
    # define MAX_NODES 1000000
    # define INFINITY 9999999
    # define NO_SELECTION -1
    # define TRUE 1
    # define FALSE 0
    # define ASSIGN 1
    # define NOT_ASSIGN 0
    # define NOT_SOLVED -1
    # define NO_VALUE -1
    # define NOT_REASSIGN 0
    # define NOT_ASSIGNED 0
    # define ASSIGNED 1
    # define SIZE 5000
```

5.2. CNP Solver

The CNP solver class is the main implementation of the branch and bound algorithm including branching, bounding, LIFO management, model initialization and etc. Below shows some piece of the solver.

```
class CnpSolver {
        int nAgents_;                  /*Number of agents*/
        int nTasks_;                   /*Number of tasks*/
        int **c ;                      /*Objective values*/
        int **r;                       /*Coefficients*/
        int *b;                        /*Capacities*/
        int *pSelectedAgent_;          /*Solution to PR problem*/
        int nSerialNo_;                /*Current node number in the queue*/
        Node* pBestNode_;              /*Best*/
        int nFeasibles;                /*Number of feasible solutions*/
        Node* pRoot_;                  /*Root node for the branch and bound tree*/
        Node* pStackHead_;             /*Head node for the LIFO stack*/
        Node *pCur_;                   /*Current node*/
        /*
        *Implementation of LIL for objective value sparse matrix
        *and Coefficients sparse matrix.
        *
        *Each row will head an array with different length
        *according to the number of non-zero cells in the row.
        */
        LILNode** ppCSparseRow_;
        LILNode** ppCSparseCol_;
        LILNode** ppRSparseRow_;

        ... omission of other members
        CnpSolver();
        CnpSolver();
        int initialize(const char* fileName );
        int solve();
private:
        int feasible_(int *nSolution);
        void reassign_();
        int pkSolver_(int &nZValue, int &nNextTask, int &nNextAgent);
        int prSolver_();
        void fanthom_(const char* str);
}
```

Other functions include a 0-1 knapsack problem solver implemented with dynamic programming approach, LIFO operation functions, etc.

CHAPTER 6. EXPERIMENT AND COMPUTATION RESULTS

To test our algorithm and demonstrate the efficiency of the algorithm, we programmed our algorithm and solved a variety of test problems. The program was written with C++ and tested on under PowerEdge 2970 with 64-bit Red Hat Enterprise Linux 5.5. We test the algorithm on randomly generated problems with different number of conflict areas, response units and team size. In the randomly generated problems, a uniform probability distribution was used to create the objective function coefficient $V_{i,j}$ values. The solution times reported in the tables below were CPU time. The solution times, the number of feasible solutions and total nodes processed are given below. Table 1 shows computation results for problem with 10 response units, 15 conflict areas and maximum team size of 5. Table 2 shows computation results for problem with 11 response units, 16 conflict areas and maximum team size of 6. Table 3 shows computation results for problem with 12 response units, 17 conflict areas and maximum team size of 7. Table 4 shows computation results for problem with 13 response units, 18 conflict areas and maximum team size of 8. Table 5 shows computation results for problem with 14 response units, 19 conflict areas and maximum team size of 9.

Table 1. Computation result for group 1

RUs	CAs	Team Sizes	Variables	Time	Feasible Solutions	Nodes
10	15	5	9200	0.035	4	49
10	15	5	9200	0.027	3	40
10	15	5	9200	0.037	4	52
10	15	5	9200	0.075	4	98
10	15	5	9200	0.041	5	57

The standard deviation of solution time and median values of the number of

Table 2. Computation result for group 2

RUs	CAs	Team Sizes	Variables	Time	Feasible Solutions	Nodes
11	16	6	14022	0.052	2	50
11	16	6	14022	0.053	3	53
11	16	6	14022	0.044	2	48
11	16	6	14022	0.05	3	50
11	16	6	14022	0.054	3	53

Table 3. Computation result for group 3

RUs	CAs	Team Sizes	Variables	Time	Feasible Solutions	Nodes
12	17	7	20424	0.424	4	227
12	17	7	20424	0.091	2	58
12	17	7	20424	0.091	2	61
12	17	7	20424	0.103	3	67
12	17	7	20424	0.135	6	82

Table 4. Computation result for group 4

RUs	CAs	Team Sizes	Variables	Time	Feasible Solutions	Nodes
13	18	8	28700	0.175	4	76
13	18	8	28700	0.141	1	66
13	18	8	28700	0.166	3	76
13	18	8	28700	0.161	3	72
13	18	8	28700	0.218	5	93

feasible solutions and total nodes are also given in Table 6. Based on the computational results, the computation time of our algorithm is relatively stable for each problem size.

In order to investigating the scalability of the algorithm, we also tested the algorithm on problems that vary widely on the problem size. The test was conducted

Table 5. Computation result for group 5

RUs	CAs	Team Sizes	Variables	Time	Feasible Solutions	Nodes
14	19	9	39168	0.62	4	172
14	19	9	39168	0.332	3	89
14	19	9	39168	0.304	2	81
14	19	9	39168	0.36	4	93
14	19	9	39168	0.404	6	105

Table 6. Statistics for five experiment groups

RUs	CAs	T Sizes	Variables	Max	Min	Avg	STD	Slts Avg	Nodes Avg
10	15	5	9200	0.076	0.027	0.043	0.038	4	59
11	16	6	14022	0.053	0.044	0.05	0.039	2	51
12	17	7	20424	0.427	0.091	0.169	0.133	3	99
13	18	8	28700	0.213	0.138	0.171	0.112	3	77
14	19	9	39168	0.614	0.299	0.397	0.264	3	108

on problems with response units, conflict areas and team sizes all different. The computation results was listed in Table 7. Figure 3 shows the scalability chart for problems listed in Table 7.

The last experiment was carried out on problems with different number of response unit that can be dismissed. We generated two groups of problems each with the same number of response units, conflict areas and team size. One group has half of the response units dismissible and another group has no response units dismissible. Table 8 shows computation results for problems with half of the response unites dismissible. Table 9 shows computation results for problems with no response unites dismissible.

Figure 4 shows the scalability chart for problems listed in Table 8 and Table 9. Based on the computation results, the solver can efficiently solve the CNP with

Table 7. Computation results for problems with different number of CA, RU and team size

RUs	CAs	Team Sizes	Variables	Time
10	15	5	9200	0.025
11	16	6	14022	0.117
12	17	7	20424	0.093
13	18	8	28700	0.229
14	19	9	39168	0.384
15	20	10	52170	0.458
16	21	11	68072	1.96
17	22	12	87264	2.4
18	23	13	110160	33.144
19	24	14	137198	41.593

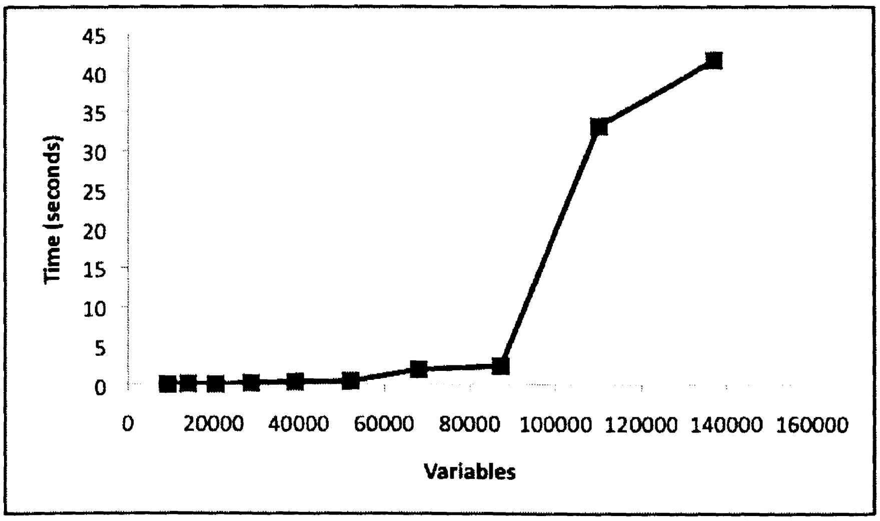


Figure 3. Scalability chart for problems with different number of CA, RU and team size

Table 8. Computation results for problems with none dismissible response units

RUs	CAs	Team Sizes	Variables	Time
7	10	5	4154	0.191
7	11	5	4964	0.199
7	12	5	5846	0.104
7	13	5	6800	0.276
7	14	5	7826	0.739
7	15	5	8924	1.71
7	16	5	10094	4.158
7	17	5	11336	7.943
7	18	5	12650	18.805
7	19	5	14036	40.645

Table 9. Computation results for problems with half of the responsible unit dismissible

RUs	CAs	Team Sizes	Variables	Time
7	10	5	4154	0.161
7	11	5	4964	0.025
7	12	5	5846	0.135
7	13	5	6800	0.196
7	14	5	7826	0.132
7	15	5	8924	0.074
7	16	5	10094	0.362
7	17	5	11336	0.477
7	18	5	12650	1.236
7	19	5	14036	11.186

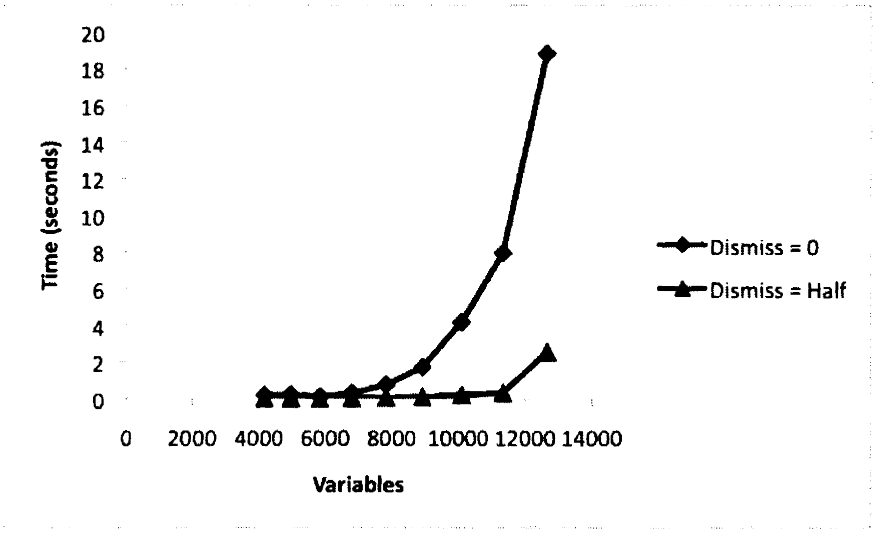


Figure 4. Scalability chart for problems with different dismissible RUs

a large range of problem size.

CHAPTER 7. CONCLUSION AND FUTURE WORK

In this report, we first presented a novel approach to model the Conflict Neutralization Problem as a Generalized Assignment Problem. The mathematical model of the CNP was presented and described. Upon examination of the CNP model, there is no existing algorithm can be applied directly for solving the problem. Therefore, we proposed in this report that the CNP could be modeled as a Generalized Assignment Problem, for which efficient algorithms exist. In the tableau representation of CNP to GAP, tasks and agents of GAP are generalized and represented as columns and rows. Constraints of the GAP model stipulate the constraints of the CNP, hence the solution to this GAP model is also a valid solution of the CNP.

Existing computer codes are not capable of processing the special structure for our CNP model and few computer codes are available to offer modeling flexibility for the problem. Also, the commercial mixed integer linear programming codes are too general to be efficient on large sized problems like the CNP. In Ross and Soland's work, they programmed and tested their algorithms in FORTRAN. Their code could efficiently solve problems with up to 4,000 variables. Later they solved the GAP model of the P-median problem, which is similar to our GAP model for the CNP. The code could solve up to 1,000 variables. From their study we can estimate that their algorithm can solve similar sized CNP problems, that is far from enough for solving the CNP. We implemented the branch and bound algorithm of Ross. To further improve the performance of the algorithm, we applied special structures in our program. The tableau representation of the CNP model shows a large amount of inadmissible cells. We can surely give large coefficients to bypass the selection of these cells. In this study, we prefer to use special LILs to avoid the processing of these inadmissible cells. In this way we greatly reduced the processing time. The computation results showed that our algorithm can efficiently solve Conflict Neutralization Problems with

4000 variables. We then tested the algorithm on problems with a large range of size, the solver can still work efficiently for problems with up to 9000 variables with none of the response units can be dismissed, which is the most intensive scenario. For each problem size, the solution times of our algorithm are relatively stable.

This report is a preliminary work on modeling and solving the Conflict Neutralization Problem. Later research can be done with some dynamic CNP scenarios simulation based on the solver implemented in this study.

REFERENCES

[1] Maria Albareda-Sambola, Maarten H. van der Vlerk, and Elena Fernndez, *Exact solutions to a class of stochastic generalized assignment problems*, European Journal of Operational Research **173** (2006), no. 2, 465–487.

[2] John Arquilla and David Ronfeldt, *Networks and netwars the future of terror, crime, and militancy*, (2002).

[3] V. Balachandran, *An integer generalized transportation problem for optimal job assignment in computer networks.*, Operations Research **24** (1976), no. 4, 742–759.

[4] J.E. Beasley and P.C. Chu, *A genetic algorithm for the generalized assignment problem*, Computers and Operations Research **24** (1997), no. 1, 17–23.

[5] Dirk G. Cattrysse and Luck N. Van Wassenhove, *A survey of algorithms for the generalized assignment problem*, European Journal of Operational Research **11** (1992), no. 2, 260–272.

[6] Juan A. Daz and Elena Fernandez, *A tabu search heuristic for the generalized assignment problem*, European Journal of Operational Research **132** (2001), 22–38.

[7] M.L. Fisher and Jaikumar, *A generalized assignment heuristic for the large scale vehicle routing*, Networks **11** (1981), no. 2, 109–124.

[8] A.M. Geoffrion, *Lagrangean relaxation for integer programming*, Mathematical Programming Studies **2** (2004), 82114.

[9] J.B. Mazzola, *Generalized assignment with nonlinear capacity interaction*, Management Science **35** (1989), 923941.

[10] A.W. Neebe and M.R. Rao, *An algorithm for the fixedcharge assigning users to sources problem*, Journal of the Operational Research Society **34** (1983), 11071113.

[11] G. Terry Ross and Richard M. Soland, *A branch and bound algorithm for the generalized assignment problem*, Mathematical Programming **8** (1975), 91–103, 10.1007/BF01580430.

[12] G. Terry Ross and Richard M Soland, *Modeling facility location problems as generalized assignment problems*, Management Science **24** (1977), 345–357.

[13] V. Srinivasan and G.L. Thompson, *An algorithm for assigning uses to sources in a special class of transportation problems*, Operations Research **21** (1972), 284–295.