MULTIPLE CHOICE PROGRAMMING

by

Timothy R. Hayes

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

A thesis submitted to the Faculty and Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Doctor of Philosophy Mathematical and Computer Sciences.

Golden, Colorado

Date 6-29-04

Signed: _____
Timothy R. Hayes

Approved: _____
Dr. R.E.D Woolsey
Thesis Advisor

Golden, Colorado

Date 30 June 2004

_____
Dr. Phillip R. Romig, Jr.
Dean of Graduate Studies

ABSTRACT

Multiple Choice Programming, **MCP**, is a unique heuristic developed by W.C. Healy to resolve mixed-binary linear decision scenarios containing ***binary sums*** or "***multiple choice***" constraints: sets of 0-1 variables summing to unity. Such problems occur in capital budgeting, site allocation, payload optimization, project scheduling, and a host of other practical situations.

MCP defines a set of linear **Profit Constraints** designed to force repeated use of the Linear Programming Simplex Method to satisfy binary sums. These constraints leverage a powerful concept, intuitively derived from the structure of the Simplex Tableau, that is generally found only in continuous optimization settings: the **Profit Gradient**. This gradient indicates progress in the current solution value per unit change in each binary decision variable, potentially enabling **MCP** to converge in a uniquely efficient manner. Also, rather than repeatedly adding new Profit Constraints during optimization as do other constraint-based methods, **MCP** updates previously constructed Profit Constraints. Both of these properties imply unique significance for **MCP**.

Although **MCP** has been used successfully in limited applications within the oil refining industry, the method is not presently known to be either finite or convergent. Consequently it seems, **MCP** has been the focus of relatively little research. This is despite the fact that **MCP**'s gradient-based approach is quite unique within its space, and in light of the fact that MCP modifies existing constraints during optimization rather than repeatedly adding new ones.

After reviewing relevant literature and Healy's original algorithm, I

- present **MCP** non-convergence theory,
- extend **MCP** to converge finitely in pure binary space,
- present preliminary benchmarking results, and
- conclude that enhanced **MCP** may be quite useful in practice.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I am, frankly, very grateful for the merciful assistance of God in the completion of this research. I have prayed my way through every stage of its development and have often sensed His pleasure in enabling and guiding me. I cannot begin to think I would have completed it apart from Him.

His hand has been manifest in a number of ways, particularly in the tireless and conscientious efforts of my most esteemed advisor and mentor, Dr. Gene Woolsey, who has been a constant and delightful source of knowledge, inspiration and practical assistance in ways too numerous to detail. It is difficult for me to imagine a more valuable advisor in the field of Operations Research and I will treasure his influence in my life for years to come.

I am also very much aware of the value of numerous contributions and insights offered by Dr. Fred Glover. His timely willingness to review the manuscript, evaluate my research and offer numerous insightful comments and encouragements are very much appreciated.

Finally, I would like to acknowledge the relentless encouragement of my grandfather, Jimmy E. Hayes, who prodded me for many years to continue until I achieved success. Though recently deceased, the memory of his fatherly concern lingers warmly within me.

GLOSSARY

$\sum_{j=1}^{j=n}$	summation of **n** terms indexed by **j**, where **j** ranges from 1 to **n**
algorithm	an optimization procedure that results in an optimal solution
coefficient	a factor that is not a variable
constraint	a mathematical relationship limiting the values of certain variables
converge	to locate an optimal solution
binary	involving only two possible values or options: represented by 0 and 1
exact	resulting in an optimal solution
feasible	satisfying all constraints
finite	converge in a limited (non-infinite) number of steps
heuristic	an optimization procedure that may not result in an optimal solution
iff	if and only if
linear	involving only sums of products of numbers with variables
integer	a number with no fractional part
IP	Integer Programming, optimization with only integers
ILP	Integer Linear Programming, optimization of linear IP
LP	Linear Programming, optimization of constrained linear problems
MILP	Mixed Integer Linear Programming, optimization in linear MIP
MIP	Mixed Integer Programming, optimization with integers and non-integers
optimal	best, most profitable, least costly
optimization	attempting to find an optimal solution to a mathematical model
Simplex	a popular LP algorithm
solution	a feasible assignment of values to the variables in a mathematical problem

tableau	a structured table representing a stage in the LP Simplex method
variable	an alpha-numeric string representing an unknown quantity or value

CHAPTER 1

INTRODUCTION

General Context

There are many practical situations when selections must be made from among groups of mutually exclusive things, perhaps picking a project to do, a route to take, a switch to flip, an object to pack, a well to dig, or the next task to sequence. There is generally also a benefit obtained from making a good choice. Like a multiple choice test in school, the goal is to score well. When the number of possible choices is large and the surrounding context is relatively complex it is very difficult to identify the most profitable choices.

When the benefits and limitations relevant to such choices are quantifiable, one may represent such scenarios mathematically and use a computer to find the best answer. In the early 1960's, under intensities of a developing Cold War between major super powers, key national security concerns could actually be posed in such mathematical terms so the need to solve them intensified greatly. However, the mathematical methods that developed were painfully slow and typically only resolved very small problems.

It was during this period, in 1964, that a uniquely innovative "Multiple Choice Programming" (**MCP**) method was published by W.C. Healy. It was intuitively appealing and appeared relatively fast, but Healy did not know if **MCP** would always find the best answer, or how many steps it might take. This limitation apparently motivated most all researchers to ignore it. It is still very little understood.

The purpose of this work is to reveal the unique nature of **MCP**, show whether or not it can always find the best answer in a finite amount of time, and -- if it cannot --

enhance **MCP** so that it can. I will:

1. Explain the science of multiple choice problems and how **MCP** fits in it,
2. Give an overview of the methods used to solve such problems,
3. Explain in detail how the original **MCP** algorithm (**MCP.0**) works,
4. Explain why **MCP.0** cannot always find the best answer and then fix it (**MCP.1**) so that it always can (subject, of course, to round-off error),
5. Present initial testing results using a variety of benchmark problems, then
6. Suggest areas for further research.

Basic Concepts

I begin by defining relevant terms and notation. Consider a standard representation often used to solve complex decisions mathematically [25].

Maximize: $$\varphi = \sum_{j=1}^{j=n} P_j X_j$$

Subject to: $$\sum_{j=1}^{j=n} a_{ij} X_j \le b_i \quad X_j \ge 0, \quad i = 1...m$$

This mathematical formulation generally represents some instance or situation in which a number of decisions must be made under certain limitations in order to achieve a benefit. The first expression above is called the ***Profit Function***: a sum of the products of ***n* decision variables**, the X_j, and their respective per-unit ***profits***, the P_j. The second expression represents the ***constraints***, restrictions on the values of the decision variables. The object of the optimization strategy is to maximize the sum φ given the problem constraints, achieving as much benefit as possible by assigning values to the X_j in an optimal manner. Minimization may also be employed where φ is the ***Cost Function***, C_j are constant per-unit ***costs*** replacing the P_j, and the object is to incur as little "cost" as possible while satisfying constraining requirements. The basic structure presented above

is suitable for either approach, thus the profit/cost function is often generalized to ***Objective Function***.

There is a generally a limit to how large φ can become since the values assigned to the decision variables are limited by aspects of the context being exploited -- limitations represented in the second expression above: the ***constraints***. There is always at least one, and often a large number of these restrictive relationships ($m \geq 1$), which are limitations on resources and/or supplies available to maximize profit. Each constraint models a type of finite resource or supply required to make profit. The b_i are non-negative constants that represent finite quantities or bounds comprising the constraints, which define the amount of each resource available. Each a_{ij}, the ***technical coefficient***, is a constant representing how much of resource **i** each unit of X_j "uses up" or "consumes" when the "decision" is made to assign a value to variable X_j.

The constraints are ***linear*** functions in X_j: there are no exponents or products among the X_j so we have "Linear Programming" or **LP**. A universal restriction in **LP** is that all decision variables are non-negative ($X_j \geq 0$). It is required of the ***solution***, the values assigned to the X_j, that it be ***feasible***, i.e. the set of values assigned to the decision variables must be such that all constraining equations are satisfied. The best solution, the one that yields the most satisfactory value for φ, is the ***optimal*** solution. Finding this solution is called ***optimization***. Algorithms that theoretically guarantee an optimal result are **exact**, those that guarantee optimality in a limited number of steps are ***finite***, and those that do not guarantee an optimal result are called ***heuristics***.

If all decision variables in such a context are integers the problem is an ***Integer Linear Programming*** problem, or an **ILP**. If all decision variables are binary ($X_j \in \{0,1\}$) this is called a ***0-1 Programming*** problem. Problems containing both integer and non-integer variables are ***Mixed Integer Linear Programming*** problems (**MILP**).

A certain type of constraint occurring commonly in integer programming is an equality with decision variables and coefficients that are all binary and having a right

hand side of unity ($b_i = 1$). This is a ***binary sum***, or a "***multiple choice***" ***constraint***. For such constraints one must choose exactly one variable in each constraining set.

A common representation of a Linear Programming problem is the Simplex Tableau in **Figure 1**.

MAX		X_1	X_2	X_3	...	X_k	...	X_n	
Basis	profit	p_{x1}	p_{x2}	p_{x3}	...	p_{xk}	...	p_{xn}	RHS
s_1	p_{s1}	a_{11}	a_{12}	a_{13}	...	a_{1k}	...	a_{1n}	b_1
s_2	p_{s2}	a_{21}	a_{22}	a_{23}	...	a_{2k}	...	a_{2n}	b_2
s_3	p_{s3}	a_{31}	a_{32}	a_{33}	...	a_{3k}	...	a_{3n}	b_3
...	...	...	...	...	...	...	...	...	...
s_i	p_{si}	a_{i1}	a_{i2}	a_{i3}	...	a_{ik}	...	a_{in}	b_i
...	...	...	...	...	...	...	...	...	...
s_m	p_{sm}	a_{m1}	a_{m2}	a_{m3}	...	a_{mk}	...	a_{mn}	b_m
reduced costs=>		a_{01}	a_{02}	a_{03}	...	a_{0k}	...	a_{0n}	φ

Figure 1. Simplex Tableau

Maximization (**max**) or minimization (**min**) is indicated in the upper left corner of the tableau. The variable names X_j are along the top in columns 1 to **n** and also in the left-most column, the ***basis*** or current solution (sack variables added to make all constraints equalities), with respective profits adjacent. The problem constraints are next to the basic profits in rows 1 to m, with right-hand-side values b_i in the right-most column (**RHS**). The current solution value, φ, is in the lower right corner. The bottom row, row zero, contains reduced profits (**RP**) corresponding to variables in their respective columns, which is the amount of increase in the objective function value that would be gained by introducing one unit of the variable in that column into the solution (the ***basis***). ***Pivoting*** a variable into the basis exchanges a column variable with one of the basis variables and updates the values in the tableau, the right hand sides and the current solution φ to reflect the new basis solution. The ***Simplex Method*** is a procedure for repeatedly selecting appropriate variables to pivot into the tableau. This method is an exact algorithm because

it locates the optimal solution for any Linear Programming problem in a finite number of steps. **Figure 1** is my convention for representing an **MCP** problem.

MCP: What is it?

Multiple Choice Programming, **MCP**, is an algorithm initially developed by W.C. Healy to optimally resolve mixed-binary linear problems containing "multiple choice" constraints... hence its name. Such problems occur frequently in oil refining (Healy's business), as well as in capital budgeting [14], site allocation [19], payload optimization, project scheduling and a host of other formulations. The algorithm employs the Simplex Method mentioned above, a standard optimization technique designed for unrestricted (non-integer) linear settings and defined in many elementary texts of Operations Research [25]. Healy's work was published in Management Science [10] in 1964, early in the development of integer programming theory, but without proof of convergence. This lack of proof of convergence may have severely limited its academic acceptance even though **MCP** suggested a unique general approach to **MILP** problems.

Now that I have explained a bit of the overall context, and provided a sense of how **MCP** fits within this context, I will consider other relevant classes of optimization methods before exploring **MCP** in detail.

CHAPTER 2

LITERATURE REVIEW

Having defined the general context of **MCP**, I will now briefly review and compare other current types of methods generally available to solve multiple choice problems.

Since **MCP** was designed to handle the "Multiple Choice" problem, which is in the 0-1 Linear Programming space, a strictly binary linear space, I will limit my discussion of current optimization methods to this space. While it is true that **MCP** solves problems where only a subset of the decisions are multiple choice decisions, MCP does not actually contribute anything of interest outside of the "multiple choice" part of the problem, so focusing on purely multiple-choice problems will more conveniently facilitate and focus my presentation of **MCP** in all of its detail.

Exact 0-1 Linear Programming optimization algorithms may be generally classified as follows:

1. Those that implicitly enumerate all possible combinations of integer variables.
2. Those that repeatedly modify the problem boundary (constraints).

Branch and Bound

The first classification given above describes the classic Branch and Bound approach as it is applied to 0-1 LP. The sundry algorithms, known as *implicit enumeration* techniques [4], effectively reduce to a single principle: efficiently eliminate

potential solutions by defining bounds on the objective function value. Included in this category are recently developed Basis Reduction strategies such as that of Wang [20], Dynamic Programming algorithms such as DPS from Martello and Toth [11], and the classic implicit enumeration strategy of E. Balas, who, along with Glover, did much of the early work in this field [25].

These types of algorithms vary in complexity and utility, but are fundamentally working on the same basic principle: develop "search trees" of potentially optimal combinations of variables and prune/eliminate the trees when their potential falls below any known optimum. To illustrate, the classic approach of Balas is considered in detail.

The implicit enumeration algorithm of Balas begins by defining a 0-1 LP minimization problem where all objective function cost coefficients are non-negative. (To reach this condition from the more general 0-1 LP formulation, replace any binary X_j with $1\text{-}X_j$ when its objective function coefficient C_j is either negative in a minimization problem or positive in a maximization problem.)

Let me continue by defining a number of necessary concepts:

1) **S**: a partial solution, initialized to the null set
2) **Z***: the best known solution value
3) **V**: the set of constraints violated when only the partial solution **S** is considered.
4) $\mathbf{F_p}$: the value of the cost function when only the partial solution **S** is considered
5) **T**: the set of candidate variables for the optimal solution:

In considering these concepts **Z***, $\mathbf{F_p}$, and **V** are intuitive in any Branch and Bound approach. **Z*** can be provided by some heuristic or known solution; in many cases all X_j can be set to unity to determine an upper bound on the minimum solution, or **Z*** can be set to infinity. $\mathbf{F_p}$ and **V**, the partial solution value and the set of violated constraints, will initially be zero and null, respectively.

The sets **S** and **T** are of special interest and deserve further investigation. **T** is naturally a set of variables not in the partial solution **S** that may be feasibly considered for membership in **S**. They must have an objective function coefficient not greater than the difference between the known optimal value and the current partial solution. They must also assist in satisfying a constraint that is currently being violated by the partial solution **S**. This implies that all variables in **T** have a positive coefficient in some constraint in **V**.

The real key to the Balas algorithm, where any Branch and Bound algorithm is ultimately defined, is in defining the partial solution **S**. There must be some approach to building the solution (***Branching***), and some means to track variables that have been eliminated from the current solution (***Bounding***). Balas uses the set **S** for both purposes.

Branching and ***Bounding*** refers to a graphical picture of what happens in implicit enumeration in a binary problem. Consider the following diagram, **Figure 2**, illustrating

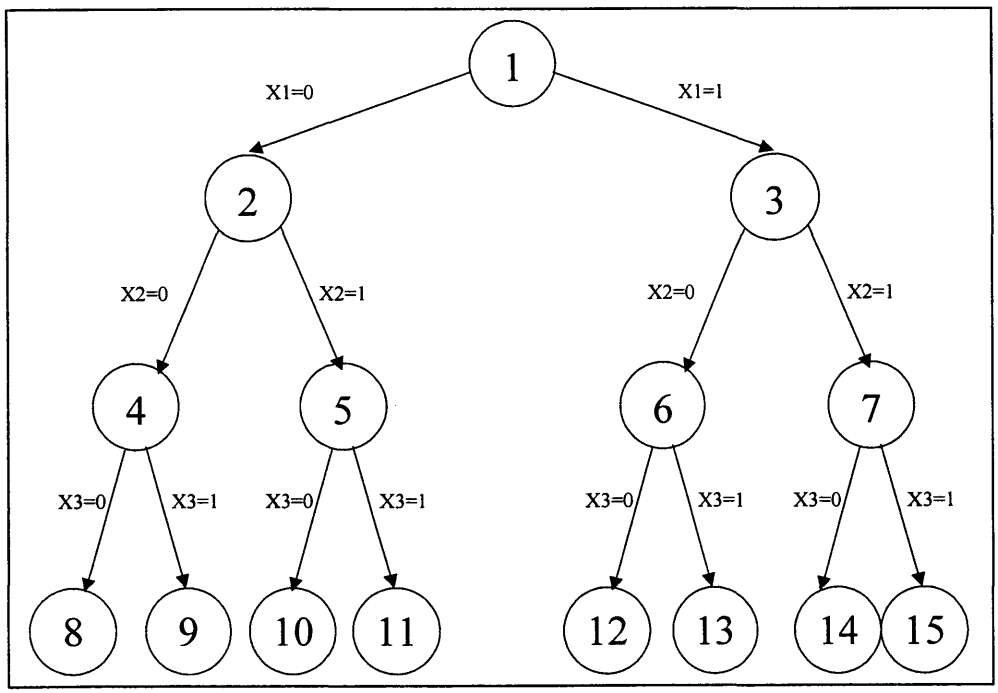

Figure 2. The Branch and Bound Search Tree

the "search tree" for a binary optimization problem in three variables: X1, X3, and X3.

The numbered circles are "nodes:" decision points where the values of the variables are set. The lines are the "branches," indicating a path through the "tree," or which variable value is to be decided next. The "tree" is "traversed" starting at the top node, fixing the value of the first variable, and moving down the branches to the next node. At each node, a decision is made about the value of a variable, the partial objective function is evaluated, and then the next variable is found. When a "leaf" node is reached (nodes 8 to 15) the objective function value is known since all variables have values. In this example, there are 8 of these terminal "leaf" nodes, representing the $2^3 = 8$ possible settings for the three variables. If at any node the current Objective Function value is already greater than the known feasible solution, the remainder of the tree from that node to any of its leaves can be "pruned" and discarded since none of these leaves can represent an optimal solution.

The guarantee of optimality in any Branch and Bound method hinges upon two properties:

1. No branch is discarded until it cannot contain a better solution than the current solution.
2. Every remaining node must be explored in searching for the optimum.

Truncating large branches often, quickly "pruning" the tree, is the secret to any Branch and Bound technique, allowing it to find the optimal answer efficiently. The faster this pruning is done, the more efficiently the optimal solution is located since fewer nodes need to be searched. Searching any significant proportion of the total number of possible combinations is generally infeasible since this number is generally huge. Even in the relatively simple binary context of 0-1 programming the number of search tree paths is $2^{\mathbf{n}}$ for problems in **n** variables. For a small problem in 50 variables we have a number of search combinations on the order of 10^{15}. Searching one million leaf nodes per second could result in a program run time of 35 years. Things are much worse for problems in

ILP, moving from binary space to integer space, and considerably more difficult in generalized MILP. Unless the pruning of the tree in a Branch and Bound algorithm is spectacularly efficient, computation time will be impractical. To date, no such spectacular method has been defined; run times for all Branch and Bound algorithms are generally impractical on real-world problems.

Balas traverses the search tree by looking at relationships between certain sets: **S**, **T**, and **V**. The ordered set **S** is built by adding to **S** (on the right) the variable that is not already in **S** having the greatest "Bang-for-Buck" measure, generally the greatest coefficient sum. **V**, the set of violated constraints, is then determined based on the elements of **S**. **T**, the set of feasible variables available to complete the partial solution, is then determined based on **V**. The set **S** is updated when both **T** and **V** are not empty and the violated constraints in **V** could be satisfied by selecting some of the variables in **T** to complete the partial solution defined in **S**.

When **T** is empty there are no more variables that can be added to the partial solution **S** to improve the current solution. This defines a "bound" condition at the node of the tree chosen by the existing partial solution in **S**: a branch is truncated, "cut off," or "pruned." This state requires the set **S** to be updated: the least desirable element in the solution based on the chosen "Bang-for-Buck" measure, that being the right-most positive element in **S**, is negated to indicate that it is removed from the solution. Then all of the variables to its right are dropped from **S** and made available again for placement in **T**. The negated element itself stays in **S** and remains unavailable for membership in **T** until it is freed up by the negation of an earlier element in the set, preventing any further exploration of its values. Consequently, the section of the tree defining its value in the context of the preceding values in the search path creating the bound is removed from further consideration by the algorithm: it is "pruned" from the search tree.

Thus **S** is built as an ordered set, adding variables with the greatest coefficient sum (from constraints and the Objective Function: a basic "Bang-For-Buck" measure) to

the right of any members previously added to **S**. Balas tracks these eliminated variables by keeping their complement (negative) in **S** in the order that they were added to **S**. Thus, the current position and signs of the members of the set **S** define the current branch of the total search tree that is being explored. Negated variables indicate truncated branches of the tree defined by positive members to its left. In other words, a variable is not permanently eliminated from the search space until its negative appears in **S** with no positive members to its left.

Whenever **V** is empty there are no violated constraints, implying that a new feasible solution has been obtained. Adding further variables to **S** in this condition would only increase the value of the objective function **Z*** so the tree can be "pruned," or truncated at that node. As in the condition when **T** is empty, "pruning" implies updating **S**: the rightmost positive element of **S** is negated and all elements to its right are dropped from **S,** making them available again for placement in **T**. When all of the members of **S** are negative and **T** is empty, the entire tree has been traversed and the algorithm terminates with an optimal **Z***.

Before beginning the Balas algorithm, it is common sense to first check whether a null solution is feasible. This would yield the best possible objective function value since all of the objective function coefficients are non-negative. Once the null solution has been eliminated, the flowchart in **Figure 3** represents the flow of the Balas algorithm.

Although the foregoing may seem a bit simplistic, it comprises the essence of any Branch and Bound technique. Sophistication may be obtained in the condition used for building the partial solution, in deciding when to truncate a branch, and in the efficiency of the comparisons. This simple algorithm of Balas may itself be further enhanced by examining constraints in **V** in the order of their perceived tightness on the optimal solution, reducing search time when deciding if **V** is feasible in **T**. Selecting a more efficient "Bang-for-Buck" metric in building **S**, based on known properties of the variables, may result in larger values for **Z*** earlier in the algorithm and more efficient

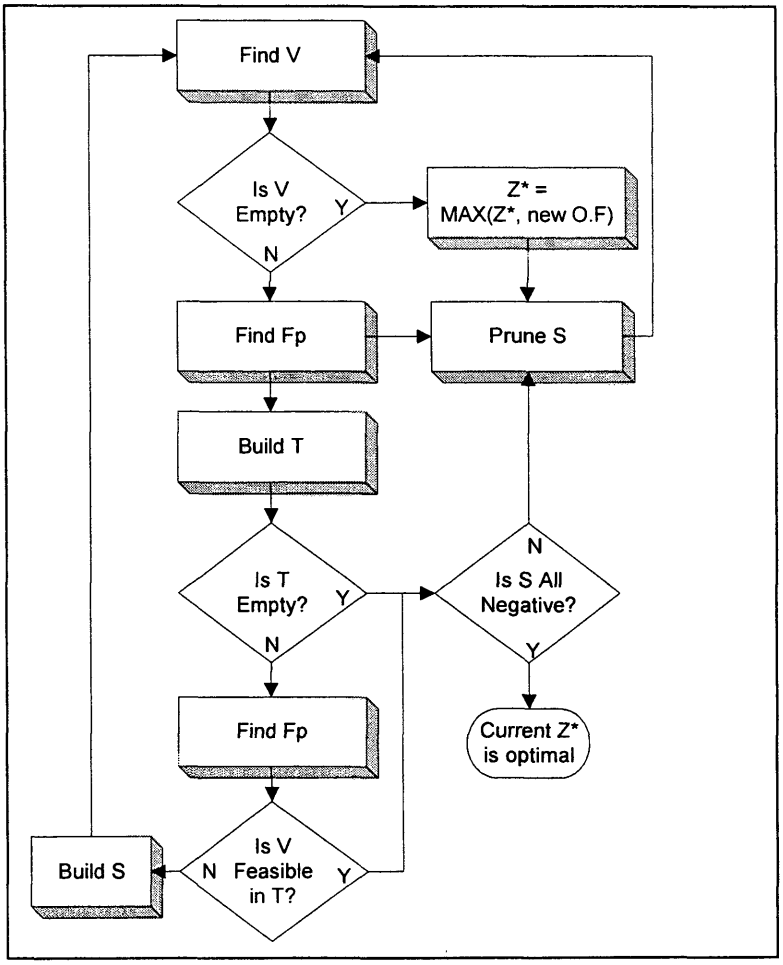

Figure 3. Balas Flowchart

"pruning." A great deal of sophistication is available in many of the current algorithms in these areas, but the underlying structure is always the same. In any case, the isolated utility of any Branch and Bound algorithm is quite limited due to the vastness of the problem size, as noted earlier. Something beyond brute force node examination must generally be considered in practice, or one must turn from exact algorithms to heuristics... settle for "good" instead of "best." Departure from the exploration of isolated combinations of variable settings while retaining optimality implies modification of the problem space: the Cutting Plane.

Cutting Plane Methods

In the other of the two broad classes of exact 0-1 Programming algorithms, the class involving the reduction of the problem space, there are two possibilities:

1. Constraints may be added repeatedly during the course of the algorithm
2. Existing constraints may be recursively modified

The first approach to constraint modification identifies a class of "cutting plane" methods that add constraints, or "cuts," to eliminate non-integer solutions. Gomory's Method of Integer forms [8] is a classic representation of this approach.

The Method of Integer Forms finds its strength in constraints derived from non-integer solutions to the Linear Programming relaxation. First, the LP is solved without binary restrictions, and then non-integer components of the LP solution are used to generate additional constraints that will reduce the search space but not exclude an integer solution.

Any treatment of this approach introduces notation to indicate "largest integer in" in order to construct the cut. The use of square brackets is convenient, where **[A]** is the largest integer **N** that is ≤ **A**. It is evident that $[a_j] \leq a_j$ for any real number a_j. It is also true for integer X_j that:

$$\sum_{j=1}^{j=n} a_j X_j = b \Rightarrow \sum_{j=1}^{j=n} [a_j] X_j \leq b \qquad \textbf{(eq. 2.1)}$$

and that $\sum_{j=1}^{j=n} [a_j] X_j$ is a sum of products of integers. Therefore $\sum_{j=1}^{j=n} [a_j] X_j$ is also an integer. Since by definition $[b]$ is the largest integer $\leq b$, and since $\sum_{j=1}^{j=n} [a_j] X_j$ is an integer $\leq b$, we must also have

$$\sum_{j=1}^{j=n} [a_j] X_j \leq [b] \qquad \textbf{(eq. 2.2)}$$

I may, as in standard simplex methodology, introduce a slack s_j to get

$$\sum_{j=1}^{j=n}[a_j]X_j + S_j = [b] \qquad \textbf{(eq. 2.3)}$$

I may also define the fractional part of a_j as f_j, and of b as f, and write

$$\sum_{j=1}^{j=n} a_j X_j \le b \Rightarrow \sum_{j=1}^{j=n}[a_j + f_j]X_j \le [b] + f \qquad \textbf{(eq. 2.4)}$$

Subtracting the right hand side of **eq. 2.4** from **eq. 2.3** produces the **Gomory Cut**:

Gomory Cut $$\sum_{j=1}^{j=n} -f_j X_j + S_j \le -f \qquad \textbf{(eq. 2.5)}$$

Trivially, all I have shown is that the sum of the fractional parts of the coefficients in any integer constraint must not be less than the fractional part of the right hand side. The Gomory Cut will be satisfied by any integer solution since any such solution satisfies **eq. 2.2**. This means that the cut, while eliminating fractional solutions to an ILP, will never eliminate an integer solution. This property can be used to reduce the search space during ILP optimization.

When a Gomory cut is added to the final simplex tableau the value of the slack variable **s** is negative. This implies that some portion of the original space is now infeasible; it has been "cut off" from the search space. The Dual Simplex method is invoked to return to primal feasibility, and integrality is re-checked. If non-integer valued variables persist then another cut is made and the process is repeated.

The fact that the Gomory Cut does not exclude an integer optimum does not, for certain, guarantee that repeated application of the cut will result in an integer solution in a finite number of steps. The choice of the variable used to generate the cut must be made deliberately in order to be certain of convergence [8].In order to guarantee convergence, a certain lexicographic condition must also be satisfied by the simplex tableau at the time a cut is made, and care must be taken to maintain this condition.

Like the Branch and Bound algorithms, the efficiency of cutting plane methods is

also quite limited in practice, but for different reasons. Using cuts to constrain the search space often results in very small "slices" being removed from the search space with each cut, implying very small incremental steps being taken toward the optimal solution. Further, with each new constraint, the problem size becomes larger which increases the number of calculations necessary to complete succeeding steps. This often causes the algorithm to converge very slowly. Finally, the combination of these two weaknesses exposes the algorithm to round-off error since computers can only handle limited numerical precision. Small inaccuracies are therefore inevitably introduced into the model during resolution that often cause the algorithm to fail.

Hybrid Methods

Before discussing the last category, constraints that are recursively modified, it is well to note that recent research in 0-1 Programming has focused on combining strengths of the Branch and Bound and the concept of a cut, in an approach generally referred to (naturally) as ***Branch and Cut***. The approach of Balas and Ceria [2] is an example of this method. Without going into lengthy detail, as this type of approach really is a combination of the two general integer programming approaches reviewed, suffice it to say that the search space is partitioned up into a number of regions (***Branching***) and optimized with a series of successively stronger LP relaxations. In each region polyhedral ***cuts*** (constraints) are constructed to drive the LP solutions on the sub-regions to integrality. The solutions to the resulting sub-problems are used to eliminate sub-regions (***Bounding***) or to further divide them, depending on whether they could potentially contain the optimal solution or not. The cuts employed are significantly different in scope than those developed by Gomory in that they are valid across the primary region even though they restrict the sub-region. The diversity among this class of algorithms lies in the nature of how the cut is formed and the relationships of the cut to the Branch and

Bound criteria. Further work in this area looks promising [3], but it would appear from the inherent design in such methods that the general Cutting Plane weaknesses will apply to these methods as well.

Generalized Gradients

In the last category, in which are techniques that recursively modify existing constraints to locate binary optima, we have the work of W.C. Healy. Multiple Choice Programming, **MCP**, is the only known method that modifies existing constraints without repeatedly adding new constraints. This property implies unique significance for **MCP**, though it is not without unique difficulties.

Healy's original publication plainly noted the limited theoretical foundation of **MCP.0**. Convergence was not known at the time of publication, and finiteness was also open to question. Healy published the algorithm with the stated hope that it, "may provoke additional investigation by others" [10]. He was not personally interested in continuing that investigation himself, and he did not.

Given the unique properties of the algorithm, subsequent work in applying and developing **MCP.0** has been minimal; there are no known publications that extend the theory of **MCP.0**. In fact, there is only one significant reference to **MCP.0** known in the literature [14]: Rousseau, a student of Woolsey, applied **MCP.0** to capital budgeting problems but experienced both inefficiency and non-convergence.

Rousseau presented a straightforward procedure for reformulating general capital budgeting problems into a format that could be solved by **MCP.0**. Essentially, Rousseau added proxy variables to complete the unique **MCP** binary sum constraints required of the **MCP** formulation whenever binary sum sets were not present. He ran seventeen test problems, of which fourteen were solved correctly by **MCP.0**.

For the remaining three test problems that did not converge, Rousseau was unable

to determine the cause of the non-convergence behavior, concluding: "The problem appears to be with the LP solver in the code. For problems that require a large number of matrix inversions, it appears that round-off error continues to compound until such a point that the correct variables are no longer in the basis."

While Rousseau is generally correct in this statement for large LP problems, these particular problems are quite small and, in my opinion, his non-convergence should not be due to round-off error. Indeed, I have determined that **MCP.0** demonstrates legitimate non-convergence behavior in each of these three problems, and will show this for one of these problems shortly.

Rousseau is the only reference to Multiple Choice Programming available in the literature that retains any theoretical substance. Convergence has not been shown, nor has any proof been published showing that **MCP.0** is non-convergent. Finiteness has also been entirely ignored. Without these basic properties there has been little interest in applying Healy's work in practice. As a result, Multiple Choice Programming remains an obscure and dusty relic of early integer programming science, and it is apparent that no business value is being currently obtained from it.

In summary, comparing the theoretical strengths and weaknesses of best in class Branch and Bound and Cutting Plane methods with **MCP**, consider the table in **Figure 4**.

Property	Balas	Gomory	MCP
Convergent	Good	Good	?
Size	Good	Poor	Good
Speed	Very Poor	Poor	Good
Round-Off Error	Good	Poor	Good

Figure 4. Theoretical Method Comparison

Each of the methods discussed is theoretically convergent, except perhaps **MCP**. Since **MCP** does not repeatedly add constraints like Gomory it should manage problem size well. The gradient-based nature of **MCP** also indicates it might be fast; **MCP** constraints may be more efficient than Gomory cuts.

There are no other known approaches to discrete optimization fundamentally based upon a gradient. While the use of gradients is common in continuous optimization scenarios, it is unheard of in discrete environments. Given such a peculiar potential strength, the general importance of this class of problems and the apparent weaknesses of existing algorithms a full exploration and development of this **MCP** algorithm is easily justified. The uniqueness of **MCP**, in its inherent algorithmic elegance and implied potential strengths, has therefore inspired the following research.

CHAPTER 3

MULTIPLE CHOICE PROGRAMMING

OVERVIEW OF MCP.0

Now that I have defined the general context of **MCP** and briefly reviewed and compared relevant best-in-class methods, I will explain in detail exactly how **MCP** works.

Often, the standard technique for resolving linearly constrained continuous problems, Linear Programming (LP), is crudely used to solve problems with integer restrictions because it is relatively fast in comparison to IP algorithms. However, LP does not have any means to guarantee conformance to the integer constraints and will typically violate them with fractional results. Practitioners are either reduced to rounding the LP solution to the nearest integer as needed and hoping for the best, which can be disappointing [14], or extending the algorithm to accommodate integer constraints. If there were some way to efficiently constrain the LP to satisfy the binary restrictions, it might be quite useful in practice. This is precisely the nature of **MCP**.

Essentially, **MCP** defines a set of linear constraints, the ***Profit Constraints***, which eliminate LP non-integer solutions but which will never eliminate the optimal integer solution. If an initial LP solution (**trial 0**) is infeasible due to violating integer constraints, then **MCP** Profit Constraints are positioned to eliminate this solution and the LP is run again (**trial 1**). If the resulting solution in this trial is infeasible, the Profit Constraints are modified with results from the previous trial and the LP is rerun (**trial 2**) to eliminate the last solution. This process continues repeatedly until either an LP solution is found that satisfies (or perhaps is apparently "close enough" to satisfying) the integer constraints and

is therefore optimal, or the Profit Constraints fail to eliminate the last non-integer solution.

Binary Sum

MCP optimizes mixed-binary linear systems containing ***binary sums***.

Binary Sum $$\sum_{j=1}^{j=n} a_{ij} X_j = 1 \quad a_{ij}, X_j \in \{0,1\}$$ **(eq. 3.1)**

Eq. 3.1 is called a ***binary sum***. It is equality constraint **i** composed of **n** binary variables X_j having binary coefficients a_{ij} and a right hand side of unity. Each $a_{ij} = 1$ implies that X_j is a ***member*** of binary sum **i**. The binary sum is a fundamental component of **MCP** formulation: every binary variable must be a member of at least one binary sum. The name "Multiple Choice Programming" is derived from this type of constraint, which requires the selection of exactly one ("multiple choice") member X_j in each binary sum **i** to be part of the optimal solution.

Implicitly, by definition, there will be at least two members in each well-posed binary sum (else, eliminate the binary sum and its single member from the problem since the member is a constant equal to unity). If a binary decision variable is not contained in a binary sum constraint, one is added consisting of two members: the binary variable and a corresponding proxy (dummy) variable.

Profit Gradient

A second fundamental concept of **MCP** is the Profit Gradient γ_j^t. The Profit Gradient is the basis for optimization in **MCP** and is developed at the end of a non-convergent LP trial **t** to determine how constraints should be modified to move toward optimality in the next trial. The Profit Gradient is defined for each binary variable X_j from

the final LP simplex tableau in trial **t** (by convention a maximization problem, with similar definitions for minimization) as follows:

Profit Gradient $\gamma_j^t = -a_{0k}$ for non-basic X_j in column **k** **(eq. 3.2)**

$\gamma_j^t = \min(a_{0k} / a_{ik} < 0)$ for basic X_j in row **i** , X_h in col **k** $\notin E_j$

The Profit Gradient γ_j^t for trial **t** is thus defined uniquely under two conditions: when variable X_j is not basic and when X_j is basic (where E_j is the ***exclusion set*** of X_j, all variables mutually exclusive of X_j).

For each non-basic binary decision variable X_j in trial **t** the Profit Gradient γ_j^t is simply the minimum deterioration (amount of loss, the ***negation*** of the amount of profit) of the profit function value that would be caused by pivoting one unit of X_j into the basis. This property of the Profit Gradient is derived from the definition of a_{0k}, the reduced profit in column **k**, which is the difference between P_k, the profit generated by one unit of X_k, and the sum of the products of the technical coefficients in column **k** with their corresponding basic profit values: $a_{0k} = P_k - \sum_{i=1}^{i=m} a_{ik} P_i$ ($a_{0k} \leq 0$ in the final maximization tableau).

For ***basic*** variable X_j in row **i**, γ_j^t is the smallest non-negative ratio that can be formed from negative technical coefficients in the row (the a_{ik}) and their respective reduced profits (the a_{0k}). Restricting ratios to negative a_{ik} implies that ratios will be formed from coefficients corresponding only to non-basic variables since the a_{ik} for basic variables are non-negative (0 or 1, thus often dropped from the tableau).

Restricting gradient ratios to those where a_{ik} is negative also implies γ_j^t represents change only from variables that would cause an ***increase*** in X_j if they were brought into the solution. This follows from the definition of the technical coefficient a_{ik}: the amount of X_j that is "given up" or "lost" -- the amount of reduction in X_j implied by bringing in

one unit of X_k to replace it. This guarantees non-negativity in the Profit Gradient ratios since all reduced profits a_{0k} in a final simplex tableau are non-positive.

When there are no non-negative entries in the row of a basic variable, it is evident that pivoting in any other variable will not increase the value of the basic variable. Under this condition, the gradient is set to a very large number: **M**. When the value of such a variable is unity it is plain why the variable may not be increased: it is binary and it is already at its maximum value. However, when the variable is less then unity and cannot be increased, it may simply be dropped from the problem: setting this variable to unity in the optimal solution is infeasible due to problem constraints.

Exclusion Set At this point, to minimize the tendency to form non-positive gradients due to the structure of the binary sum, a peripheral concept must be introduced: E_j, the exclusion set for X_j. Essentially, when calculating γ_j^t it is desirable to only consider those variables X_k which would potentially cause an increase in X_j in the optimal solution. By definition, this would exclude any variables that are co-members with X_j in a binary sum since including any such variable in the solution would cause X_j to leave the solution due to the constraining property of the binary sum. Healy, in developing **MCP.0**, did note experimentally that if this consideration were not made when constructing the gradient **MCP.0** would often fail to converge. To overcome this limitation in the performance of the algorithm he introduced for each variable X_j the concept of its "***exclusion set***:" the set **j** of variables being mutually exclusive of X_j. The presence of any member of the exclusion set in the optimal solution implies the absence of all other members of the set due to the effect of the binary sums. Simply, ***exactly*** one member of any particular binary sum may be present in the optimal solution. If a binary variable appears in multiple binary sums, then all of the members of any such binary sum become members of this variable's exclusion set. Therefore, as in the last expression in **eq. 3.2**, when constructing the

gradient, any negative aik is ignored by definition when the corresponding Xk belongs to the exclusion set of Xj.

"Minimum Rate of Change" A result deduced from the definition of γ_j^t is that it represents the "minimum rate ***of change***" of the profit function value φ with respect to the variable X_j. The phrase is a delightful -- and indeed surprising -- one to come across in the field of Integer Programming. It so happens that one may view γ_j^t much like a partial derivative in the multi-variable differential calculus: γ_j^t is exactly the ***minimum rate of change*** of φ with respect to X_j at X_j^t, and it is a constant. This gives the Profit Gradient a convenient property [10]: γ_j^t is the greatest lower bound to the rate of change of φ with respect to the variable X_j at trial **t** or:

$$\gamma_j^t \leq \frac{(\varphi^t - \varphi)}{(1 - X_j^t)} \qquad \textbf{(eq. 3.3)}$$

where φ^t is the value of the objective function at some **MCP** trial **t**, φ is the optimal objective function value, and X_j^t is the value of the j[th] decision variable X_j in the same **MCP** trial **t**.

Suppose X_j is basic and in row **i** of the final maximization simplex tableau in trial **t**. If some X_k in column **k** is introduced to replace some other basic X_h, then X_j will increase iff (*if and only if*) $a_{ik}^t < 0$. The resulting decrease in profit φ is exactly $b_i^t a_{ik}^t / a_{hk}^t$ and the ***rate*** of decrease is exactly a_{ik}^t / a_{hk}^t. Since γ_j^t is the minimum among all such ratios (and since exclusion set members are omitted) it follows that γ_j^t is exactly the minimum rate of change of φ with respect to X_j. If it so happens that all $a_{ik}^t < 0$, it follows that X_j cannot be increased and I assign an infinite value to γ_j^t.

When X_j is non-basic in the final tableau of trial **t**, it is well known that $-a_{0k}$ represents exactly the rate of change of φ with respect to X_j.

Thus γ_j^t is non-negative, and it is -- by definition -- the largest deterioration guaranteed to occur in the current profit function per unit increase in X_j. This is a result derived from commonly known properties of the LP... and it is certainly unique in this IP context. If the current solution is not optimal, if there is some binary variable X_j whose value is not zero or one, then some other variable must be brought into the solution. The Profit Gradient γ_j^t, like the partial derivative of calculus, represents the largest guaranteed change in the value of the profit function per unit increase in X_j. This property can be used to reduce the search space.

Profit Bound

Building on the properties of the Binary Sum and the Profit Constraint, a third **MCP** concept is developed in order to uniquely constrain a linear space to an integer space: the Profit Bound: φ_j^t. As with γ_j^t, the Profit Bound φ_j^t is calculated for each binary variable X_j at the end of a non-convergent LP trial **t** and it is derived from the Profit Gradient.

Essentially φ_j^t is an upper bound on the optimal profit function value φ if X_j=1. It is defined as follows:

Profit Bound $$\varphi_j^t = \min\left(\varphi_j^{t-1},\ \varphi^t - \left(1 - X_j^t\right)\gamma_j^t\right) \qquad \textbf{(eq. 3.4)}$$

This equation states that after the LP run at trial **t**, φ_j^t is found by decreasing the current profit function value φ^t by the largest guaranteed change in the profit function value per unit change in X_j (γ_j^t) times the total change in X_j if X_j^t were forced to unity (1-X_j^t). The value X_j^t is the value of X_j in the solution of the final simplex tableau in trial **t**:

for basic X_j the value x_j^t will be in the closed interval [0,1], and x_j^t is zero for any non-basic X_j.

If the new φ_j^t is not an improvement ($\varphi_j^{t-1} \leq \varphi_j^t$), then keep the most constraining value for φ_j^t since the smallest such value ever found for X_j is the most productive Profit Bound to use. Note also that if γ_j^t for basic X_j is set to a very large number, **M**, due to not having any non-negative entries in its row of the tableau, the gradient has no effect iff $x_j^t = 1$. When $x_j^t < 1$ and γ_j^t = **M** the Profit Bound becomes –**M** and X_j consequently drops from the problem (since $\varphi > -\mathbf{M}$ in a well-posed LP problem).

This φ_j^t at trial **t** is thus an upper bound on the largest value of the profit function if the binary variable X_j is unity in the optimal solution.

Profit Constraint

The following step is natural once the best φ_j^t is known for each binary decision variable X_j at trial **t**: form a new LP constraint that combines the effect of all Profit Bounds upon the optimal solution. For each binary sum constraint **i**, a Profit Constraint is defined as follows:

Profit Constraint i
$$\sum_{j=1}^{j=n} P_j X_j \leq \sum_{j=1}^{j=n} a_{ij} \varphi_j^t X_j \qquad \textbf{(eq. 3.5)}$$

Here, P_j represents the "profit" (or cost, in minimization) obtained from finding X_j=1 in the optimal solution. I obtain the current profit function value at trial **t** from this relationship: i.e.

$$\sum_{j=1}^{j=n} P_j X_j^t = \varphi^t$$

The left side of the **eq. 3.5** is the optimal solution φ. The right side reduces to the

sum of the products of the binary variables and their respective Profit Bounds that are all members of the same binary sum **i** ($a_{ij} = 1$ for member X_j, else $a_{ij} = 0$). This implies that a unique Profit Constraint can be constructed from each unique binary sum to reduce the search space. Since exactly one of the binary variables in any Profit Constraint is unity in the optimal solution, and since each $\varphi_j^t \geq \varphi$ if $X_j = 1$, I have the result that the sum of the products of the variables in the binary sum and their respective Profit Bounds forms an upper bound on φ.

What is perhaps not so obvious is that the Profit Constraint actually does actively constrain the search space when positive Profit Gradients exist. Furthermore, the last non-integer LP solution will violate this constraint so long as there has been an improvement in a Profit Bound associated with an optimal decision variable. If positive gradients exist for only non-optimal variables, the search space will be reduced but the optimal objective function may not change. Regardless, so long as positive Profit Gradients exist, including Profit Constraints in the LP formulation will significantly reduce the search space. LP can be run repeatedly, updating the Profit Bounds with each trial, to reduce the search space over a number of successive trials. This is the key result, implying an algorithm to progressively constrain the search space until there is zero improvement in each and every Profit Bound, suggesting (but not *implying*, as we shall see) convergence of the algorithm. This is the very essence of **MCP** and the concept merits a thorough justification.

Justification Simply, I wish to derive a constraint, the Profit Constraint, which bounds the search space by considering, for all binary decision variables, the smallest guaranteed change that would occur in the current profit function value if each decision variable were part of the optimal integer solution.

To begin with, as Healy reasoned [10], we know (in maximization) that $\varphi \leq \varphi^t$.

For any binary sum constraint we know that $\Sigma a_{ij} X_j = \Sigma a_{ij} X_j^t$ (Σ is over all j = 1...n) since both the (unknown) optimal solution and the current LP solution satisfy all binary sum constraints (which are of the form $\Sigma a_{ij} X_j = 1$). I can combine these two concepts into an inequality representing what happens to the current solution φ^t when $X_j^t \leq 1$ is feasibly introduced into the optimal solution (set equal to unity).

Consider the following inequality for a certain binary sum constraint **i**:

$$\sum a_{ij} X_j^t + \Delta \leq \sum a_{ij} X_j + \varphi^t - \varphi \qquad \textbf{(eq. 3.6)}$$

In review, X_j is binary, X_j^t is the value of X_j in the non-convergent LP solution of trial **t** (where at least one X_j^t is not zero or unity), a_{ij} is a binary constant associated with X_j in the binary sum constraint, φ^t is the LP solution at trial **t,** and φ is the unknown optimal solution. Let each summation only include members of the binary sum (i.e. a_j =1).

Building on $\Sigma a_{ij} X_j^t = \Sigma a_{ij} X_j$, I derive **eq 3.6** by forming a small non-negative number Δ, not greater than the difference between the current LP solution φ^t and the optimal solution φ. I add Δ and its upper bound $\varphi^t - \varphi$ to opposites sides, forming an inequality. $\varphi \leq \varphi^t$ implies both $\varphi^t - \varphi \geq 0$ and $\Delta \geq 0$, as per my definition. Also, when φ is strictly $< \varphi^t$ I require that Δ is also strictly positive, and visa versa. In other words, with positive Δ there is potential for improvement in the profit function because we have not yet arrived at an optimal objective function value and at least one X_j^t is non-integer.

When Δ is positive there is a conflict in the constraint between the variables X_j and φ (remember that a_{ij}, φ^t and X_j^t are constants) so there will be a trade-off between adjusting φ and the X_j in satisfying the constraint while **MCP** moves toward optimality. Since LP is trying to maximize φ it will force changes in the X_j in order to satisfy the equation instead of adjusting φ. Thus, adding to the formulation a constraint equivalent to

eq. 3.6 will significantly constrain the search space when Δ is positive. What remains is:

1. Find convenient representation for Δ to show it is positive with a non-integer solution.
2. Eliminate the a_{ij}.

Define γ_j^t, the gradient at trial **t**, as we have previously: the minimum guaranteed reduction in the profit function value at trial **t** per unit change in X_j implied by including X_j in the solution. For $X_j^t = 1$ let $\gamma_j^t = \infty$. If $X_j^t < 1$ we know (from **eq. 3.3**) that γ_j^t is a lower bound to $(\varphi^t - \varphi) / (1 - X_j^t)$: the change in φ per unit change in $\mathbf{X}_j$. Since Δ is a lower bound to $\varphi^t - \varphi$ (**eq. 3.6**) a convenient representation for Δ that includes the gradient is established from the expression $(1 - X_j^t)(\varphi^t - \varphi) / (1 - X_j^t)$ as follows:

Definition of Delta $$\Delta = (1 - X_j^t)\gamma_j^t \qquad \textbf{(eq. 3.7)}$$

This is a valid definition for Δ since $\gamma_j^t \le (\varphi^t - \varphi) / (1 - X_j^t)$, from **eq 3.3**, implies $(1 - X_j^t)\gamma_j^t \le \varphi^t - \varphi$, which in turn implies my requirement that $\Delta \le \varphi^t - \varphi$. Then for each X_j I find an equation in Δ that can be solved in terms of X_j and a_{ij}: simply set each X_j in **eq. 3.6** in turn to unity. For each X_j I get the following:

Equation in Delta $$\Sigma a_{ij} X_j^t + \Delta - a_{ij} = (1 - X_j^t)\gamma_j^t \qquad \textbf{(eq. 3.8)}$$

The left side is similar to the left side of **eq. 3.6**, with the $-a_{ij}$ coming from what is left of $\Sigma a_{ij} X_j$ on the right side of **eq. 3.6** after setting $X_k = 0$ for all $\mathbf{k} \neq \mathbf{j}$. The right side is obtained by multiplying the remainder of the right side of **eq 3.6** (being $\varphi^t - \varphi$) by $1 = (1 - X_j^t)/(1 - X_j^t)$ and then substituting γ_j^t as a lower bound to the expression $(\varphi^t - \varphi)/(1 - X_j^t)$. Since I desire an equation in Δ and I have by definition $\gamma_j^t \le (\varphi^t - \varphi) / (1 - X_j^t)$, and from **eq. 3.6** we have $\Sigma a_{ij} X_j^t + \Delta - a_{ij} \le \varphi^t - \varphi$, I can safely remove the inequality and have **eq. 3.8** as an equality. I solve this for Δ.

Remembering that $\Sigma a_{ij} X_j^t$ is unity for any binary sum, for each X_j multiply both sides of **eq. 3.8** by the term $a_{ij} X_j^t$ from the binary sum and combine resulting equations.

$$a_{i1} X_1^t \Sigma a_{ij} X_j^t + \Delta a_{i1} X_1^t - (a_{i1})^2 X_1^t = (1 - X_1^t) \gamma_1^t X_1^t$$

$$a_{i2} X_2^t \Sigma a_{ij} X_j^t + \Delta a_{i2} X_2^t - (a_{i2})^2 X_2^t = (1 - X_2^t) \gamma_2^t X_2^t$$

$$\cdots$$

$$+ \; a_{in} X_n^t \Sigma a_{ij} X_j^t + \Delta a_{in} X_n^t - (a_{in})^2 X_n^t = (1 - X_n^t) \gamma_n^t X_n^t$$

$$(\Sigma a_{ij} X_j^t)^2 + \Delta \Sigma a_{ij} X_j^t - \Sigma (a_{ij})^2 X_j^t = \Sigma (1 - X_j^t) \gamma_j^t X_j^t$$

Since a_{ij} is binary and $\Sigma a_{ij} X_j^t = 1$ the first and third terms on the left cancel, leaving Δ as follows:

Delta in X_j^t, γ_j^t $$\Delta = \Sigma X_j^t (1 - X_j^t) \gamma_j^t \qquad \textbf{(eq. 3.9)}$$

Observe from **eq. 3.9** that:

$$\Delta = 0 \text{ iff for all } \mathbf{j} \text{ either } X_j^t \in \{0,1\} \text{ or } \gamma_j^t = 0$$

Since I would like $\Delta > 0$ for all non-integer X_j^t it is essential that $\gamma_j^t > 0$ unless $X_j^t \in \{0,1\}$ (motivating the "exclusion set" concept... and, conveniently, providing a key to discovering non-convergence criteria). I now have a convenient representation for Δ.

To eliminate the a_{ij}, I let $\alpha = \Sigma a_{ij} X_j^t = \Sigma a_{ij} X_j$ and reconstruct **eq. 3.6** by subtracting $\Sigma \alpha X_j^t$ from the left side and $\Sigma \alpha X_j$ from the right side and combining summations as follows:

$$\Sigma (a_{ij} - \alpha) X_j^t + \Delta \le \Sigma (a_{ij} - \alpha) X_j + \varphi^t - \varphi \qquad \textbf{(eq. 3.10)}$$

From **eq. 3.8** (by rearranging its terms a bit) I know that $a_{ij} - \alpha = \Delta - (1 - X_j^t) \gamma_j^t$, so I can eliminate the a_{ij} by substitution in **eq. 3.10** as follows:

$$\Sigma (\Delta - (1 - X_j^t) \gamma_j^t) X_j^t + \Delta \le \Sigma (\Delta - (1 - X_j^t) \gamma_j^t) X_j + \varphi^t - \varphi$$

Multiplying through by X_j^t and splitting summations I have:

$$\Sigma\ \Delta\ X_j^t - \Sigma(1-X_j^t)\gamma_j^t\ X_j^t\ +\ \Delta \le\ \Sigma\Delta X_j\ -\ \Sigma(1-X_j^t)\gamma_j^t X_j + \varphi^t - \varphi$$

Since $\Sigma\Delta\, X_j^t = \Delta\Sigma\, X_j^t = \Delta$ I have:

$$\Delta\quad -\Sigma(1-X_j^t)\gamma_j^t\ X_j^t\ +\Delta \le\quad \Delta\quad -\ \Sigma(1-X_j^t)\gamma_j^t X_j + \varphi^t - \varphi$$

Subtracting Δ from both sides I have:

$$-\Sigma(1-X_j^t)\gamma_j^t\ X_j^t\ +\Delta \le\qquad -\Sigma(1-X_j^t)\gamma_j^t X_j + \varphi^t - \varphi$$

Solving for Δ I have: $\Delta \le\quad \Sigma(1-X_j^t)\gamma_j^t)\,X_j^t\ -\Sigma(1-X_j^t)\gamma_j^t X_j + \varphi^t - \varphi$

From **eq. 3.9** I have: $\Delta \le\qquad \Delta\qquad -\Sigma(1-X_j^t)\gamma_j^t X_j + \varphi^t - \varphi$

Subtracting Δ from each side and rearranging terms I get:

$$\varphi \le\qquad \varphi^t - \Sigma(1-X_j^t)\gamma_j^t\ X_j$$

Remembering that $\Sigma X_j = 1$ I have: $\varphi \le \Sigma X_j\varphi^t - \Sigma(1-X_j^t)\gamma_j^t\ X_j$

Combining summations I have:

$$\varphi \le \Sigma X_j(\varphi^t\ -(1-X_j^t)\gamma_j^t) \qquad \textbf{(eq. 3.11)}$$

Conveniently, it so happens that the coefficient of X_j in this equation, $(\varphi^t - (1-X_j^t)\gamma_j^t)$, is equivalent to the definition of φ_j^t in **eq 3.4**. I now have, at last:

$$\varphi \le \Sigma\varphi_j^t X_j \qquad \textbf{(eq 3.12)}$$

In other words, the optimal solution is upper-bounded by the sum of the products of the binary decision variables which are members of binary sum **i** with their respective Profit Bounds. By definition, I may represent φ as $\Sigma P_j X_j$ and reintroduce a_{ij} as a coefficient on the $\varphi_j^t X_j$ in binary sum **i**. Doing so allows a tidy representation of the new Profit Constraint corresponding to any binary sum **i**:

$$\Sigma P_j X_j - \Sigma a_{ij}\varphi_j^t X_j \le 0 \quad \text{or}$$

$$\Sigma(P_j - a_{ij}\varphi_j^t)X_j \le 0 \qquad \textbf{(eq. 3.13)}$$

From **eq. 3.11** it is evident that when a positive Profit Gradient is found for some non-binary valued variable that is zero in the optimal solution, the value of the corresponding Profit Bound is irrelevant. However, positive gradients guarantee progress toward the optimum when the corresponding variables are unity in the optimal solution and are non-integer-valued in the current trial (implying key insight into the nature of the optimum when progress is not made). In such case, the current LP solution will not satisfy **eq. 3.13**; adding the Profit Bound constraint to the Simplex Tableau and rerunning the LP guarantees progress toward the optimum solution. The fact that we are using a gradient based on the properties of the simplex tableau indicates that this progress might be efficient.

Feasibility Adding such constraints to reduce the search space is fruitless unless there is some way to guarantee that the optimal solution will never be excluded. Convergence cannot be guaranteed – even with the development of a profitable constraining mechanism on the search space -- if the algorithm will bypass the optimum solution in reducing the search space. The Profit Constraint must therefore retain optimal feasibility.

Suppose, as Healy reasons [10], there are binary sums of X_j's, Y_j's, Z_j's, ..., the optimal solution is $X_a=Y_b=Z_c=\ldots=1$, and that the optimal solution value has been eliminated from the search space.

Then there is a first trial τ in which a Profit Constraint is formed that is too tight (say the X_j's are the tightest) which eliminates the optimal solution φ from the search space. It happens for the first time in the following trial $\tau+1$ that

$$\varphi^t \geq \varphi > \varphi^{t-1} \qquad \textbf{(eq. 3.14)}$$

Then, from **eq. 3.12**, the definition of the Profit Constraint, I have $\varphi > \Sigma \varphi_j^t X_j$, where the eliminated optimal solution (including, among other values, $X_a = 1$, and

remaining X_j's = 0) causes the tight Profit Constraint to fail in trial τ +1; implying (from **eq. 3.12**) that the Profit Bound for X_a, φ_a^t, violates the optimal solution value:

$$\varphi_a^t < \varphi \qquad \textbf{(eq. 3.15)}$$

Since φ_a^t is the most constraining bound on the following LP trial I know that the next LP solution is the same as the limiting Profit Bound:

$$\varphi_a^t = \varphi^{t+1} \qquad \textbf{(eq. 3.16)}$$

I also know, from the definition of τ, that the optimal solution is violated first by the Profit Bound for X_j at trial τ, so prior Profit Bounds for X_j (or Y_b, or Z_c, or ...) are not less than the optimal objective function value:

$$\varphi_a^{t-1} \geq \varphi \qquad \textbf{(eq. 3.17)}$$

Another consequence concerns the erroneous X_a Profit Bound φ_a^t, which by definition in **eq. 3.4** is $\min(\varphi_a^{t-1},\ \varphi^t - (1 - X_a^t)\gamma_a^t)$. Since the optimal solution value φ is not greater than the last profit bound (**eq. 3.17**), and since the optimal solution value is strictly greater than the offending profit bound (**eq. 3.15**), I have $\varphi_a^t < \varphi \leq \varphi_a^{t-1}$. So the offending profit bound is strictly less than φ_a^{t-1}. By definition:

$$\varphi_a^t = \varphi^t - (1 - X_a^t)\gamma_a^t \qquad \textbf{(eq. 3.18)}$$

We may consider two cases in which such a violation might occur: $X_a^t = 1$ and $X_a^t < 1$.

If $X_a^t = 1$ then, from **eq. 3.16** I know $\varphi_a^t = \varphi^{t+1}$, so I have $\varphi^t = \varphi^{t+1}$ and a contradiction of **eq. 3.14** implying $\varphi^t > \varphi^{t+1}$. If $X_a^t < 1$ I have $\varphi > \varphi_a^t = \varphi^t - (1 - X_a^t)\gamma_a^t$ (**eqs. 3.15, 3.18**), implying $\gamma_a^t > (\varphi^t - \varphi)/(1 - X_a^t)$ and contradicting the definition of γ_a^t.

Thus we know that the Profit Bound will never exclude the optimal integer solution. The Profit Constraint can be utilized in the simplex method even though the algorithm itself will not respect the binary nature of the variables during solution development. We have a method of repeatedly constraining the search space with new

Profit Bound information until the optimal solution is on a boundary point of the search space or positive Profit Gradients are not found.

Additionally, the fact that the current φ_j^t for any decision variable X_j is independent of the trial in which the Profit Bound was obtained allows me to ***update*** the Profit Constraints to reduce the search space from one trial to the next instead of continually adding new ones. This conveniently keeps the problem size from expanding during algorithm execution: Profit Constraints are not repeatedly added... the performance degradation experienced with other cutting plane methods is avoided. This is the substance of **MCP**: utilization of the Profit Gradient to generate the Profit Constraint.

Example 1: A Simple Illustration

No conceptual presentation such as the foregoing would be complete without an example to illustrate the concepts. Consider the simple problem given in **Figure 5**, which exposes **MCP.0** quite well.

max	X_1	X_2	X_3	X_4		
p_j	6	3	4	8		RHS
C_1	-2	0	4	2	<=	5
C_2	3	1	2	2	<=	4
C_3	1	1	1	1	<=	2

Figure 5. MCP.0 Overview Ex

No binary sum constraints are inherent in this maximization problem in four binary variables, and so are provided. The initial tableau for trial 0 is shown in **Figure 6**.

Recall from the definition of the Simplex Tableau that variable names are along the top and left with respective profits adjacent. The problem constraints are next to the basic profits, with right-hand- side values in the right-most column (**RHS**), and the initial solution value (–80) in the lower right corner. The bottom row, row zero, contains

max		X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	
	p_j	6	3	4	8	0	0	0	0	RHS
S_9	-20	1	0	0	0	1	0	0	0	1
S_{10}	-20	0	1	0	0	0	1	0	0	1
S_{11}	-20	0	0	1	0	0	0	1	0	1
S_{12}	-20	0	0	0	1	0	0	0	1	1
S_{13}	-20	3	1	2	2	0	0	0	0	5
S_{14}	-20	1	1	1	1	0	0	0	0	4
S_{15}	-20	-2	0	4	2	0	0	0	0	2
	RP	26	23	24	28	20	20	20	20	-80

Figure 6. MCP.0 Overview Ex: Trial 0, Initial Tableau

educed profits (**RP**) corresponding to variables in their respective columns, which represent the amount of increase in the objective function value that would be gained by introducing one unit of the variable in that column into the basis.

Note that the original constraints are in the last three rows of the interior portion of the tableau (with basic slacks S_{13} to S_{15}), and are preceded by four additional binary sum constraints, one for each primary decision variable. Four proxy binary variables (X_5 to X_8) have been added to complete the binary sums. Slack variables on the binary sum equalities (S_9 to S_{12}) are given a negatively large profit (–20) to ensure their exclusion from the final solution. After performing the simplex method to complete trial 0, the final tableau is given in **Figure 7**.

max		S_{13}	S_{14}	X_3	S_{12}	S_9	S_{10}	S_{11}	X_8	
	p_j	-20	-20	4	-20	-20	-20	-20	0	RHS
X_5	-20	-½	½	-½	½	1	0	0	½	½
X_6	-20	½	-1½	-½	½	0	1	0	½	½
X_7	-20	0	0	1	0	0	0	1	0	1
X_4	8	0	0	0	1	0	0	0	1	1
X_1	6	½	-½	½	-½	0	0	0	-½	½
X_2	3	-½	1½	½	-½	0	0	0	-½	½
S_{15}	-20	1	-1	5	-3	0	0	0	-3	4
	RP	-1½	-1½	-½	-23½	-20	-20	-20	-3½	12½

Figure 7. MCP.0 Overview Ex: Trial 0, Final Tableau

It is clear that the resulting LP solution is not optimal since there are fractional values (**RHS**) for binary variables X_1, X_2, X_5, and X_6, which are all equal to ½. The Profit Gradients, Profit Bounds, and key Profit Constraint coefficients (where the corresponding variable is a member of the respective binary sum) are determined from the final tableau in **Figure 8** for each decision variable as follows:

To begin the next trial, trial 1, the Profit Constraints are listed in the same order as their corresponding binary sum counterparts (slack S_9 in the first binary sum, slack S_{16} in the corresponding Profit Constraint, etc.) in **Figure 8** below.

X_n	**Profit Gradient Calculation**	**Profit Gradient**	**Profit Bound Calculation**	**Profit Bound**	**Profit Constraint Coefficient**
X_1	Min(-3.5/-.5, -1.5/-.5, -23.5/-.5)	3	12.5-(1-.5)3	11	6 -11 = -5
X_2	Min(-3.5/-.5, -1.5/-.5 ,-23.5/-.5)	3	12.5-(1-.5)3	11	3 -11 = -8
X_3	-(-0.5)	0.5	12.5-(1-0).5	12	4 -12 = -8
X_4	No valid gradient; no $a_{ij} < 0$	**M**	12.5-(1-1)**M**	12.5	8 -12.5 = -4.5
X_5	Min(-.5/-.5, -1.5/-.5)	1	12.5-(1-.5)1	12	0 -12 = -12
X_6	Min(-1.5/-1.5)	1	12.5-(1-.5)1	12	0 -12 = -12
X_7	No valid gradient; no $a_{ij} < 0$	**M**	12.5-(1-1)**M**	12.5	0 -12.5 = -12.5
X_8	-(-3.5)	3.5	12.5-(1-0)3.5	9	0 - 9 = -9

Figure 8. MCP.0 Overview Ex: Profit Constraint Calculations

The initial tableau for trial 1 is as follows:

max		X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	
	p_j	6	3	4	8	0	0	0	0	RHS
S_9	-20	1	0	0	0	1	0	0	0	1
S_{10}	-20	0	1	0	0	0	1	0	0	1
S_{11}	-20	0	0	1	0	0	0	1	0	1
S_{12}	-20	0	0	0	1	0	0	0	1	1
S_{13}	-20	3	1	2	2	0	0	0	0	4
S_{14}	-20	1	1	1	1	0	0	0	0	2
S_{15}	-20	-2	0	4	2	0	0	0	0	5
S_{16}	-20	-5	3	4	8	-12	0	0	0	0
S_{17}	-20	6	-8	4	8	0	-12	0	0	0
S_{18}	-20	6	3	-8	8	0	0	-12.5	0	0
S_{19}	-20	6	3	4	-4.5	0	0	0	-9	0
	RP	26	23	24	28	20	20	20	20	-80

Figure 9. MCP.0 Overview Ex: Trial 1 Initial Tableau

The final tableau of trial 1 is given in **Figure 10**:

max		S_9	S_{10}	S_{15}	S_{16}	S_{17}	S_{11}	S_{12}	X_8	
	p_j	-20	-20	-20	-20	-20	-20	-20	0	RHS
X_1	6	4	-3	-0.0833	0.333	-0.25	0	-0.5	-0.5	0.0833
X_2	3	-8	9	-0.0833	-0.667	0.75	0	-0.5	-0.5	0.0833
S_{18}	-20	-9	-2.25	-0.1875	-0.75	-0.1875	12.5	-0.125	-0.125	0.1875
X_7	0	-2	1.5	-0.2083	-0.167	0.125	1	0.75	0.75	0.2083
S_{13}	-20	-8	3	-0.0833	-0.667	0.25	0	1.5	1.5	0.0833
S_{14}	-20	2	-4.5	-0.0417	0.167	-0.375	0	0.75	0.75	0.0417
X_3	4	2	-1.5	0.20833	0.167	-0.125	0	-0.75	-0.75	0.7917
X_4	8	0	0	0	0	0	0	1	1	1.0000
X_5	0	-3	3	0.08333	-0.333	0.25	0	0.5	0.5	0.9167
X_6	0	8	-8	0.08333	0.667	-0.75	0	0.5	0.5	0.9167
S_{19}	-20	-8	-3	-0.0833	-0.667	-0.25	0	12	3	0.5833
	RP	-28	-23	-0.0833	-0.667	-0.25	-20	-20.5	-0.5	11.917

Figure 10. MCP.0 Overview Ex: Trial 1 Final Tableau

Improvement is noted in the new profit function value (φ^1=11.916667) due to the Profit Constraints, which have restricted the solution space and eliminated the initial non-integer solution of φ^0=12.5. Fractional values still exist for binary decision variables, so an integer solution has not been reached and the algorithm continues, repeatedly calculating Profit Gradients, Profit Bounds, and revising the Profit Constraints with the new Profit Bound values and employing the LP simplex method, resulting in the solution progress shown in **Figure 11**.

t	$\varphi^{(t)}$	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
1	11.916667	0.0833	0.0833	0.7917	1.0000	0.9167	0.9167	0.2083	0.0000
2	11.731302	0.0249	0.1773	0.7625	1.0000	0.9751	0.8227	0.2375	0.0000
3	11.587084	0.1130	0.0000	0.8329	0.9472	0.8870	1.0000	0.1671	0.0528
4	11.401768	0.3077	0.3157	0.1522	1.0000	0.6923	0.6843	0.8478	0.0000
5	11.226429	0.1436	0.3550	0.3249	1.0000	0.8564	0.6450	0.6751	0.0000
6	11.136986	0.2429	0.2948	0.1988	1.0000	0.7571	0.7052	0.8012	0.0000
7	11.076570	0.1707	0.3588	0.2440	1.0000	0.8293	0.6412	0.7560	0.0000
8	11.043031	0.2216	0.3311	0.1800	1.0000	0.7784	0.6689	0.8200	0.0000
9	11.021360	0.1859	0.4065	0.1717	1.0000	0.8141	0.5935	0.8283	0.0000
10	11.009764	0.2133	0.4203	0.1173	1.0000	0.7867	0.5797	0.8827	0.0000
11	11.003431	0.1787	0.5366	0.0803	1.0000	0.8213	0.4634	0.9197	0.0000
12	11.000925	0.1723	0.6073	0.0363	1.0000	0.8277	0.3927	0.9637	0.0000
13	11.000136	0.1067	0.7701	0.0124	1.0000	0.8933	0.2299	0.9876	0.0000
14	11.000011	0.0670	0.8632	0.0020	1.0000	0.9330	0.1368	0.9980	0.0000
15	11.00000022	0.0194	0.9610	0.0002	1.0000	0.9806	0.0390	0.9998	0.0000
16	11.0000000013	0.0058	0.9885	0.0000	1.0000	0.9942	0.0115	1.0000	0.0000
17	11.00000000000013	0.0000	0.9992	0.0000	1.0000	0.9995	0.0008	1.0000	0.0000
18	11	0.5000	0.0000	0.0000	1.0000	0.5000	1.0000	1.0000	0.0000

Figure 11. MCP.0 Overview Ex: Solution Progress

It is apparent from observing the pattern of the trial solutions, the φ^t, that **MCP.0** is converging on φ=11. In scanning down the columns containing the decision variable values and noting their progress from trial to trial, it is apparent up through trial 17 that **MCP.0** is converging on the solution X_2=X_4=X_5=X_7=1. This solution *does* yield a total

profit of 11 so it is clear that the optimal solution cannot be less than 11.

However, at trial 18, the converging pattern is broken with respect to the binary variables and a non-convergent step is taken: a non-binary alternate optimum (X_1=X_5=½, X_4=X_6=X_7=1) surfaces at φ=11. The final tableau of trial 18 is given in **Figure 12**.

max		X_2	X_3	X_8	S_{12}	S_{13}	S_{14}	S_{15}	S_{16}	
	p_j	3	4	0	-20	-20	-20	-20	-20	RHS
X_9	0	1	5.3333	-4.6667	3.6667	0	0	-4.6667	0.3333	4.0
X_{10}	0	-0.5	0	2	-5.5	0	0	2	-0.5	0.5
S_{11}	-20	0.5	0.3333	0.3333	-1.8333	0	0	0.3333	-0.1667	0.5
X_1	6	0.5	0.6667	-1.3333	1.8333	0	0	-1.3333	0.1667	0.5
S_{17}	-20	0	0	0	-11	11	0	0	-1	0.0
S_{18}	-20	0	0.0610	0	-11	0	11	0	-1	0.0
X_7	0	0	1	0	0	0	1	0	0	1.0
X_4	8	0	0	1	0	0	0	1	0	1.0
X_5	0	-0.5	-0.6667	1.3333	-0.8333	0	0	1.3333	-0.1667	0.5
X_6	0	1	0	0	0	1	0	0	0	1.0
S_{19}	-20	0	0	2	-11	0	0	11	-1	0.0
	RP	0	0	0	-20	-20	-20	-20	-1	11.0000

Figure 12. MCP.0 Overview Ex: Final Tableau

The Profit Gradients for each decision variable in trial 18 are given in **Figure 13** below. No positive Profit Gradients exist for any $X_j \neq 1$. Forming a constraint with the resulting Profit Bounds is meaningless... such constraints will not further constrain the search space. Though **MCP.0** has located the optimal profit function value, it cannot construct any more Profit Constraints and terminates. At no point in the progress of the algorithm did **MCP.0** locate an integer solution. This is an example of convergence failure for **MCP.0**, and satisfies my initial research goal, which is to show that **MCP.0** is not convergent.

X_n	Profit Gradient Calculation	Profit Gradient	Profit Bound Calculation	Profit Bound
X_1	Min(-20/-1.33, 0/-1.33)	0	11.0-(1-.5)0	11
X_2	0	0	11.0-(1-0)0	11
X_3	0	0	11.0-(1-0)0	11
X_4	No valid gradient; no $a_{ij} < 0$	**M**	11.0-(1-1)**M**	11
X_5	Min(0/-.5, 0/-.7, -20/-.8, -1/-.17)	0	11.0-(1-.5)0	11
X_6	No valid gradient; no $a_{ij} < 0$	M	11.0-(1-1)**M**	11
X_7	No valid gradient; no $a_{ij} < 0$	**M**	11.0-(1-1)**M**	11
X_8	0	0	11.0-(1-0)0	11

Figure 13. MCP.0 Overview Ex: Profit Constraint Calculations

Finiteness / Convergence

In completing the algorithm, there should be some stopping criteria... the algorithm should be finite. If all of the values of the decision variables are binary at the end of a simplex trial, this -- by definition -- is an optimal solution and one can terminate the algorithm under this condition. Clearly, as per this last example, **MCP.0** does not imply this type of convergence. Furthermore, it is possible that accumulated round-off error may mask an optimal solution in large problems. Evidently, requiring solution values to be exactly binary is impractical.

Healy, in dealing with this subject, noting that **MCP.0** does appear to approach an optimal solution, simply introduced the standard convergence concept of an epsilon neighborhood. He defined convergence as the state when all binary variables are within some arbitrarily small distance from their binary constraint. In the above problem, a neighborhood of 0.1 implies convergence at trial 15.

In trying to salvage **MCP.0** with this convention, Healy does introduce ambiguity

into the algorithm's convergence behavior. For example, $\varepsilon = 0.2$ in the above problem yields an **in**feasible solution of $X_3 = X_4 = X_5 = X_6 = 1$ and a super-optimal result $\varphi = 12$ in trial 3 (implicitly requiring a feasibility check). Also, a bound of 0.3 gives a feasible solution of $X_4 = X_5 = X_6 = X_7 = 1$ and a sub-optimal $\varphi = 8$ in trial 6... but then one might require that φ be within some other epsilon neighborhood of φ^t.

This approach is certainly less than an elegant: it is unclear how small ε must be. While it is "mathematically" possible to pick an infinitely small ε, it is still theoretically possible that at any particular ε rounding the decision variables to the nearest integer will cause a "jump" to an incorrect solution. This causes the algorithm to terminate prematurely without the correct solution, a solution that might be found with a "small enough" ε. There will therefore always be some positive probability of non-convergent behavior in the algorithm. Further, regardless of the choice of ε, it is certainly not yet certain that the condition noticed above, where it was impossible to construct a positive gradient, will not occur prior to approaching φ.

In light of the general importance of multiple choice optimization, the potential practical value of **MCP**, and the above inefficiencies it is important to completely understand all of the conditions under which **MCP** might fail, or become inefficient, and, if at all possible, to devise means to overcome these weaknesses.

CHAPTER 4

ENHANCING MCP

Now that I have defined the general context of **MCP,** briefly reviewed and compared best-in-class methods with **MCP,** explained in detail exactly how **MCP** works and exposed some of its limitations I will describe how to enhance **MCP** so that it will finally be useful in practice. In doing so I face three challenges:

1. I must describe when and why the Profit Constraint might fail to eliminate non-integer LP solutions – when positive profit gradients might not exist even when binary constraints are being violated, if indeed this can occur.
2. I must devise a way to resolve this condition without excluding the optimal solution.
3. I must improve the "Epsilon Neighborhood" with a more rigorous termination criterion.

   In and of themselves, each of these efforts is non-trivial. Together, they have composed a somewhat formidable impasse for those working with **MCP**.

Understanding Non-Convergence

First, let me define the concept of a "stall" in the **MCP.0** algorithm as a condition in which convergence failure occurs prior to reaching φ, where the profit constraints imply no further progress.

I know from the concept of the Profit Constraint that **MCP.0** eliminates φ^t iff $\gamma_j^t >$

0 when $X_a^t < 1$ and $X_j = 1$. Thus a stall condition is implied when $\gamma_j^t = 0$ for all $X_j^t < 1$. Also, recall the following definition:

Profit Gradient $\gamma_j^t = -a_{0k}$ for non-basic X_j in column **k** **(eq. 3.2)**

$\gamma_j^t = \min(a_{0k} / a_{ik} < 0)$ for basic X_j in row **i**

The gradient will be zero for non-basic X_j iff its $a_{0k}=0$; that is, iff X_j is a member of an alternate optimum. For basic X_j the gradient will be zero iff there is at least one $a_{ik} < 0$ where $a_{0k} = 0$, implying that the smallest ratio $a_{0k} / a_{ik} = 0$.

This condition naturally occurs, as in my last example, when the optimal value φ is reached but the solution is non-integer: that is, when there are alternate non-integer optima. When this occurs the optimal solution can be obtained by pivoting through all of the optima... though finite, this is certainly a non-trivial task. While this is technically a convergence failure for **MCP.0**, at least the optimal objective function **value** is obtained and there is a finite way to conclude the algorithm. The question then naturally arises: Can the Profit Gradients stall before reaching the optimal profit function value?

When the optimal objective function value has not been reached in trial **t**, when there is a positive difference between φ^t and φ, there must be some basic $X_j < 1$ that can be increased. There must be some technical coefficient a_{ik} that is negative – indicating that X_j can be increased by pivoting X_k into the basis. For the Profit Gradient γ_j^t to be insignificant when $X_j < 1$, a_{0k} must be 0, implying that X_j will *remain* in the solution and will be increased when X_k is brought into the basis. This also implies that there is yet another variable X_h that would be replaced by X_k when X_k does enter the basis set. If this condition holds true for all basic members of the solution less than unity in trial **t MCP.0** will stall before reaching the optimal profit function value.

This implies, for each basic, non-integer-valued member X_j, that there are at least two other variables: a basic X_h and a non-basic X_k that are members of distinct alternate optima. The non-basic variable X_k must replace X_h when it is brought into the solution

basis, and doing so must cause an increase in X_k. This is a very unique condition, generally occurring when there are sets of three or more variables that are, so to speak, interchangeable. Consider the following binary problem.

Example 2: A Minimal Non-Convergent Construct

max	X_1	X_2	X_3	X_4	X_5	X_6		
C_i	2	2	2	0	0	0		RHS
C_1	2	2	2	0	0	0	≤	3
C_2	1	0	0	1	0	0	=	1
C_3	0	1	0	0	1	0	=	1
C_4	0	0	1	0	0	1	=	1

Figure 14. Minimal Non-Convergent Construct

This problem may be solved easily by inspection, as follows:

- At most one of the variables X_1, X_2, and $X_3 = 1$ due to constraint C_1.
- The solution $X_1 = X_5 = X_6 = 1$ is feasible and therefore optimal.

The optimal objective function value is therefore 2; are three possible alternate solutions:

#	X_1	X_2	X_3	X_4	X_5	X_6
O_1	1	0	0	0	1	1
O_2	0	1	0	1	0	1
O_3	0	0	1	1	1	0

Figure 15. Example 2 Optima

The initial simplex tableau of trial **0** is:

max		X_1	X_2	X_3	X_4	X_5	X_6	
	p_j	2	2	2	0	0	0	RHS
S_7	-20	2	2	2	0	0	0	3
S_8	-20	1	0	0	1	0	0	1
S_9	-20	0	1	0	0	1	0	1
S_{10}	-20	0	0	1	0	0	1	1
	RP	22	22	22	20	20	20	-120

Figure 16. Minimal Construct: Initial Tableau

The sixth and final tableau is given in **Figure 17**.

max		S_8	S_7	X_1	S_9	S_{10}	X_6	
	p_j	-20	-20	2	-20	-20	0	RHS
X_2	2	0	0.5	1	0	-1	-1	0.5
X_3	2	0	0	0	0	1	1	1
X_4	0	1	0	1	0	0	0	1
X_5	0	0	-0.5	-1	1	1	1	0.5
	RP	-20	-21	0	-20	-20	0	3

Figure 17. Minimal Construct: 6th Tableau

The optimal LP solution at trial **0** (not trial **1,** which is, by convention, the first trial in which **MCP** constraints are active) is non-convergent (since $X_2 = X_5 = ½$) so the **MCP.0** algorithm is invoked and gradients are formed. X_1 and X_6 are non-basic so their gradients are both zero (the value in the **RP** row of the columns with X_1 and X_6 at the top). The remaining variables are basic so their gradients are formed from the ratios:

$a_{0k}/(a_{ik} < 0)$. For the gradient of X_2, variable X_6 is negative, with a reduced cost of zero. Since this is by definition the minimum possible gradient, there is no need to consider $\mathbf{S}_{10}$ (which also has a negative technical coefficient) to find the minimum ratio. Similarly, for the gradient of X_5, variable X_1 is negative and provides a minimal gradient with its zero reduced cost. X_3 and X_4 are both unity so there are no negative a_{ik}; their gradients are irrelevant and they will drop out of the Profit Bound calculation since: $(1-x_j^0)\mathbf{M} = (1-1)\mathbf{M} = 0\mathbf{M} = 0$. Therefore, of the potentially meaningful Profit Gradients... $\gamma_1^0 = \gamma_2^0 = \gamma_5^0 = \gamma_6^0 = 0$. Forming the Profit Bounds from these gradients is pointless, since zero gradients imply no effect on the profit bounds ($\varphi_j^0 = 3-(1-X_j)\times\mathbf{0} = 3$). The Profit Constraints will look like:

$$\Sigma P_j X_j \leq \Sigma 3 X_j = 3\Sigma X_j = 3$$

Such constraints will not further constrain this example problem: no further progress can be made to resolve it with **MCP.0**.

This simple example illustrates a compounded alternate optima situation in which a triplet of interchangeable variables exists. In this particular instance, regardless of the type of pivoting done to alter the final state of the tableau, exactly two of these three variables will be basic; one of these basic variables will always be unity, and the other will always be 0.5. There is no means to construct a positive gradient with **MCP.0**, so the algorithm fails to converge. Certainly, one may observe that the example was specifically constructed to achieve this. It would be interesting to know if such a condition commonly occurs in practice.

Example 3: Haldi Non-convergence

Consider **Figure 18**, a test problem presented by Haldi [8] and one of the non-convergent problems encountered by Rousseau. Though this problem appears quite

simple, non-convergent behavior is clearly evident.

max	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	
p_j	20	18	17	15	15	10	5	3	1	1	**RHS**
C	30	25	20	18	17	11	5	2	1	1	≤ 65

Figure 18. Haldi: Non-Convergent Example

An initial re-formulation is necessary to put the problem in the proper format, requiring that each variable be a member of a binary sum. A binary sum constraint and a proxy variable is added for each of the ten decision variables. I select ε=.01. The first tableau of trial 0 appears as follows:

max		X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	X_{12}	X_{13}	X_{14}	X_{15}	X_{16}	X_{17}	X_{18}	X_{19}	X_{20}	
	p_i	20	18	17	15	15	10	5	3	1	1	0	0	0	0	0	0	0	0	0	0	**RHS**
S_{21}	-200	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
S_{22}	-200	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
S_{23}	-200	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
S_{24}	-200	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
S_{25}	-200	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
S_{26}	-200	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
S_{27}	-200	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1
S_{28}	-200	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1
S_{29}	-200	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1
S_{30}	-200	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1
S_{31}	0	30	25	20	18	17	11	5	2	1	1	0	0	0	0	0	0	0	0	0	0	65
	RP	220	218	217	215	215	210	205	203	201	201	200	200	200	200	200	200	200	200	200	200	-2000

Figure 19. Haldi: Trial 0 Initial Tableau

Trial 0 concludes with an initial upper bound of 58.67 as follows:

max		S_{21}	S_{22}	S_{31}	X_1	S_{25}	X_2	S_{27}	S_{28}	S_{29}	S_{30}	S_{23}	S_{24}	S_{26}	X_{13}	X_{15}	X_{16}	X_{17}	X_{18}	X_{19}	X_{20}	
	p_i	-200	-200	0	20	-200	18	-200	-200	-200	-200	-200	-200	-200	0	0	0	0	0	0	0	RHS
X_4	15	0	0	.056	1.67	-.94	1.39	-0.28	-.11	-0.06	-0.06	-1.11	0	-0.61	-1.1	-0.94	-0.61	-0.28	-0.11	-0.06	-0.06	0.444
X_6	10	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1
X_{11}	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
X_{12}	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
X_5	15	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
X_{14}	0	0	0	-0.06	-1.67	.944	-1.39	0.28	.11	.056	.056	1.111	1	.611	1.11	.944	.611	.278	.111	.056	.056	0.556
X_7	5	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1
X_8	3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1
X_9	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1
X_{10}	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1
X_3	17	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1
	RP	-200	-200	-0.83	-5	-201	-2.83	-201	-201	-200	-200	-200	-200	-201	-.33	-0.83	-0.83	-0.83	-1.33	-0.17	-0.17	58.67

Figure 20. Haldi: Trial 0 Final Tableau

After trial 0, Profit Bounds and Profit Constraints imply the first tableau of trial 1:

max		X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	X_{12}	X_{13}	X_{14}	X_{15}	X_{16}	X_{17}	X_{18}	X_{19}	X_{20}	
	p_i	20	18	17	15	15	10	5	3	1	1	0	0	0	0	0	0	0	0	0	0	RHS
S_{21}	-200	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
S_{22}	-200	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
S_{23}	-200	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
S_{24}	-200	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
S_{25}	-200	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
S_{26}	-200	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
S_{27}	-200	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1
S_{28}	-200	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1
S_{29}	-200	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1
S_{30}	-200	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1
S_{31}	0	30	25	20	18	17	11	5	2	1	1	0	0	0	0	0	0	0	0	0	0	65
S_{32}	0	-33.7	18.0	17.0	15.0	15.0	10.0	5.0	3.0	1.0	1.0	-58.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
S_{33}	0	20.0	-37.8	17.0	15.0	15.0	10.0	5.0	3.0	1.0	1.0	0.0	-58.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
S_{34}	0	20.0	18.0	-41.7	15.0	15.0	10.0	5.0	3.0	1.0	1.0	0.0	0.0	-58.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
S_{35}	0	20.0	18.0	17.0	-43.5	15.0	10.0	5.0	3.0	1.0	1.0	0.0	0.0	0.0	-57.8	0.0	0.0	0.0	0.0	0.0	0.0	0
S_{36}	0	20.0	18.0	17.0	15.0	-43.7	10.0	5.0	3.0	1.0	1.0	0.0	0.0	0.0	0.0	-57.8	0.0	0.0	0.0	0.0	0.0	0
S_{37}	0	20.0	18.0	17.0	15.0	15.0	-48.7	5.0	3.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	-57.8	0.0	0.0	0.0	0.0	0
S_{38}	0	20.0	18.0	17.0	15.0	15.0	10.0	-53.7	3.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	-57.8	0.0	0.0	0.0	0
S_{39}	0	20.0	18.0	17.0	15.0	15.0	10.0	5.0	-55.7	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-57.3	0.0	0.0	0
S_{40}	0	20.0	18.0	17.0	15.0	15.0	10.0	5.0	3.0	-57.7	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-58.5	0.0	0
S_{41}	0	20.0	18.0	17.0	15.0	15.0	10.0	5.0	3.0	1.0	-57.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-58.5	0
	RP	220	218	217	215	215	210	205	203	201	201	200	200	200	200	200	200	200	200	200	200	-2000

Figure 21. Haldi: Trial 1 Initial Tableau

The first 30 trials resolve for the ten primary decision variables (their respective binary sum proxy counterparts are valued with the difference between these primary variables and unity) as in **Figure 22**.

t	φ^t	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
1	58.50000	0.0000	0.0000	0.5000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	58.35135	0.0000	0.0000	0.9433	0.7991	0.7770	0.8673	1.0000	1.0000	1.0000	1.0000
3	58.33286	0.0039	0.0073	0.4778	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
4	58.25354	0.0000	0.0000	0.8794	0.8615	0.8412	0.8412	1.0000	1.0000	0.3503	1.0000
5	58.16587	0.0000	0.0000	0.8934	0.8867	0.8409	0.8120	1.0000	1.0000	0.0000	0.9437
6	58.09908	0.0000	0.0000	0.9022	0.8886	0.8518	0.8249	1.0000	1.0000	0.0000	0.4067
7	58.04647	0.0000	0.0000	0.9093	0.8970	0.8538	0.8271	1.0000	1.0000	0.0000	0.0558
8	58.00446	0.0000	0.0000	0.9162	0.9013	0.8572	0.8290	0.9528	1.0000	0.0000	0.0000
9	57.97049	0.0000	0.0000	0.9221	0.9062	0.858	0.8286	0.9092	1.0000	0.0000	0.0000
10	57.94276	0.0000	0.0000	0.9278	0.9102	0.8582	0.8266	0.8753	1.0000	0.0000	0.0000
11	57.92000	0.0000	0.0000	0.9333	0.9142	0.8573	0.8229	0.8505	1.0000	0.0000	0.0000
12	57.90126	0.0000	0.0000	0.9387	0.9181	0.8555	0.8174	0.8331	1.0000	0.0000	0.0000
13	57.88581	0.0000	0.0000	0.9441	0.9222	0.8528	0.8099	0.8226	1.0000	0.0000	0.0000
14	57.87311	0.0000	0.0000	0.9495	0.9265	0.8494	0.7999	0.8186	1.0000	0.0000	0.0000
15	57.86274	0.0000	0.0000	0.9551	0.9315	0.8454	0.7870	0.8209	1.0000	0.0000	0.0000
16	57.85437	0.0000	0.0000	0.9608	0.9371	0.8407	0.7705	0.8297	1.0000	0.0000	0.0000
17	57.84774	0.0000	0.0000	0.9666	0.9438	0.8358	0.7492	0.8456	1.0000	0.0000	0.0000
18	57.84264	0.0000	0.0000	0.9727	0.9517	0.8312	0.7217	0.8692	1.0000	0.0000	0.0000
19	57.83890	0.0000	0.0000	0.9788	0.9609	0.8276	0.6868	0.9008	1.0000	0.0000	0.0000
20	57.83635	0.0000	0.0000	0.9849	0.9709	0.8265	0.6434	0.9398	1.0000	0.0000	0.0000
21	57.83477	0.0000	0.0000	0.9904	0.9808	0.8293	0.5931	0.9832	1.0000	0.0000	0.0000
22	57.83393	0.0000	0.0000	0.9947	0.9892	0.8362	0.5418	1.0000	1.0000	0.0000	0.1241
23	57.83355	0.0000	0.0000	0.9976	0.995	0.8445	0.4997	1.0000	1.0000	0.0000	0.2850
24	57.83341	0.0000	0.0000	0.9991	0.998	0.8505	0.4743	1.0000	1.0000	0.0000	0.3784
25	57.83336	0.0000	0.0000	0.9997	0.9993	0.8533	0.463	1.0000	1.0000	0.4192	0.0000
26	57.83334	0.0000	0.0000	0.9999	0.9998	0.8542	0.4592	1.0000	1.0000	0.4333	0.0000
27	57.83334	0.0000	0.0000	1.0000	0.9999	0.8599	0.4472	1.0000	1.0000	0.4648	0.0000
28	57.83333	0.0000	0.0000	1.0000	1.0000	0.8553	0.4562	1.0000	1.0000	0.0000	0.4427
29	57.83333	0.0000	0.0000	1.0000	1.0000	0.8667	0.4333	1.0000	1.0000	0.0000	0.4999
30	57.83333	0.0000	0.0000	1.0000	1.0000	0.5833	1.0000	0.8167	1.0000	0.0000	0.0000

Figure 22. Haldi: Solution Progress

A stall pattern has developed. After thirty trials, **MCP.0** stalls on a non-binary objective function solution value: φ^{30}=57.83333... or $57^5/_6$. Clearly, this is not the optimal solution value since it is not an integer (all of the profits are integers, and all of the decision variables are binary, implying φ must be an integer).

Two alternate optima are discernable in **Figure 22**: in trials 29 and 30:

1. $\mathbf{O}_1$: $X_1=X_2=X_9=0$, $X_3=X_4=X_7=X_8=1$, $X_5=.87\ldots={}^{13}/_{15}$, $\mathbf{X}_6=.43\ldots={}^{13}/_{30}$, $\mathbf{X}_{10}=.5=½$.
2. $\mathbf{O}_2$: is $X_1=X_2=X_9=X_{10}=0$, $X_3=X_4=X_6=X_8=1$, $\mathbf{X}_5=.5833\ldots={}^{7}/_{12}$ and $X_7=.8166\ldots={}^{49}/_{60}$.

Both solutions yield a profit function value of $57^5/_6$; it is apparent that **MCP.0** is unable to proceed.

Analysis of the final tableau reveals three other alternate optima, O_3, O_4 and O_5 in addition to O_1 and O_2 noted above. In all five solutions, $X_1=X_2=0$, and $X_3=X_4=X_8=1$, as in **Figure 23**.

$\mathbf{X}_n$	$\mathbf{P}_j$	$\mathbf{O}_1$	$\mathbf{O}_1\times P_j$	$\mathbf{O}_2$	$\mathbf{O}_2\times \mathbf{P}_j$	$\mathbf{O}_3$	$\mathbf{O}_3\times \mathbf{P}_j$	$\mathbf{O}_4$	$\mathbf{O}_4\times \mathbf{P}_j$	$\mathbf{O}_5$	$\mathbf{O}_5\times \mathbf{P}_j$
$\mathbf{X}_3$	17	1	17	1	17	1	17	1	17	1	17
$\mathbf{X}_4$	15	1	15	1	15	1	15	1	15	1	15
$\mathbf{X}_5$	15	${}^{13}/_{15}$	13	${}^{7}/_{12}$	$8^3/_4$	1	15	${}^{19}/_{30}$	9½	1	15
$\mathbf{X}_6$	10	${}^{13}/_{30}$	$4^1/_3$	1	10	${}^{1}/_{10}$	1	${}^{11}/_{15}$	$7^1/_3$	${}^{1}/_{6}$	$1^2/_3$
$\mathbf{X}_7$	5	1	5	${}^{49}/_{60}$	$4^1/_{12}$	1	5	1	5	1	5
$\mathbf{X}_8$	3	1	3	1	3	1	3	1	3	1	3
$\mathbf{X}_9$	1	0	0	0	0	1	1	0	0	1	1
$\mathbf{X}_{10}$	1	½	½	0	0	${}^{5}/_{6}$	${}^{5}/_{6}$	1	1	${}^{1}/_{6}$	${}^{1}/_{6}$
	φ^{30}		$57^5/_6$		$57^5/_6$		$57^5/_6$		$57^5/_6$		$57^5/_6$

Figure 23. Haldi: Alternate Optima

Besides the consistently binary variables X_1, X_2, X_3, X_4, and X_8, the five remaining variables X_5, X_6, X_7, X_9, and X_{10} are each unity in at least one of the solutions and are less than unity in at least one of the solutions. Thus, in **MCP** trial 30, when any non-binary variable is less than unity in the final tableau, there exists an alternate optimum in which that variable will increase to unity. This implies that in any tableau representing one of these five alternate solutions, in each of the rows corresponding to fractionally valued variables there is a negative technical coefficient in a column with a zero reduced profit.

This condition has caused **MCP.0** failure. **MCP.0** stalls prior to reaching φ due to the existence of a set of interchangeable non-integer-valued variables.

While it may not be clear merely from the tableau that ALL alternate sub-optima will have a similar condition, it is absolutely clear that **MCP.0** cannot proceed beyond this particular condition once it has been reached. Given that **MCP** does appear promising due to its gradient-based structure, overcoming non-convergence is highly desirable... and evidently necessary if **MCP** is to become useful.

One might consider searching through the alternate optima for a solution. As noted previously, if the optimal profit function value has been obtained, then the optimal solution exists on a boundary lattice point of the LP space: some intersection of the boundary constraints contains an optimal solution and pivoting through all of the possible combinations of alternate optima will certainly find such an optimum. However, making an efficient initial determination that the optimal profit function value has been obtained is certainly beyond the scope of the **MCP.0** algorithm as it stands, and beyond the conventional use of the simplex tableau as well: this, apparently, is not a trivial task.

If the optimal profit function value has **not** been obtained, as in our current problem, pivoting through alternate optima will certainly be fruitless. One would be searching (at length) for an integer optimum where it does not exist; this would certainly be an inefficient step. This is the case in the above example: no binary combinations of integers can give me 57 and 5/6: **MCP**.0 has not yet arrived at the optimal objective function value.

One of two options is available, either:

1. remove all but two of the alternate optima variables from each of the sets of interchangeable variables and rerun LP, or
2. make use of some other means of reducing the search space.

Selecting variables to remove from a search space is a challenge, to say the least. If, for any reason, any variable or set of variables is removed from the problem such that

all optimal integer solutions are excluded from the search space (assuming perhaps that there might be multiple alternate integer optima), then the modified algorithm certainly fails. Gradients formed of the LP resulting from this improperly constructed subset of the original space may certainly result in a non-convergent termination of the algorithm.

It is inferred from this conclusion that before any variable is removed, there must be a way to determine that the optimal solution may be obtained with the remaining variables. In general, it is impossible to make this decision as it implies knowing the solution to the problem. Certainly, as the algorithm is designed to determine the optimal solution, knowledge of the optimal solution may not be required as part of the algorithm. This approach may be discarded.

Overcoming Non-convergence: the Cutting Plane

The second approach is reasonable, and it may serve to help determine whether the optimal profit function value has been obtained. A fractional cut can be constructed to reduce the search space, Dual Simplex invoked to remove the non-integer LP solution, and Profit Gradients constructed from the resulting tableau. This will certainly reduce the search space if the optimal profit function value has not been obtained as it is known that methods such as a variant of Gomory's Fractional Forms are convergent, albeit slowly. In this case, however, much of the work has already been done by **MCP** and the cuts will only be needed under unique conditions.

Example 2: Solving the Minimal Non-Convergent Construct

In the first clearly non-convergent example given in **Figure 14**, the Minimal Non-Convergent Construct, where reaching the optimal solution is impossible irrespective of ε, a fractional cut added as a fifth constraint (**Figure 24**) resolves non-convergence in one Dual Simplex pivot (**Figure 25**).

max		S_8	S_7	X_1	S_9	S_{10}	X_6	
	P_j	-20	-20	2	-20	-20	0	RHS
X_2	2	0	½	1	0	-1	-1	½
X_3	2	0	0	0	0	1	1	1
X_4	0	1	0	1	0	0	0	1
X_5	0	0	-½	-1	1	1	1	½
S_{11}	0	0	-½	0	0	0	0	-½
	RP	-20	-1	0	-20	-20	0	3

Figure 24. Gomory Cut on Minimal Construct

max		S_8	S_{11}	X_1	S_9	S_{10}	X_6	
	P_j	-20	-20	2	-20	-20	0	RHS
X_2	2	0	1	1	0	-1	-1	0
X_3	2	0	0	0	0	1	1	1
X_4	0	1	0	1	0	0	0	1
X_5	0	0	1	-1	1	1	1	1
S_7	-20	0	-2	0	0	0	0	1
	RP	-20	-2	0	-20	-20	0	2

Figure 25. Gomory Cut Resolution of Minimal Construct

Careful application of the cutting plane methods [3] guarantees that the dual simplex pivot will eventually result in an integer solution. After the cut is added and dual simplex is performed, optimality is checked. While non-integrality persists, if at any time a Gomory cut results in a reduction of the profit function value φ^t then one thing is absolutely certain: the optimal profit function value has not been obtained at the **MCP** stall point. The Gomory cuts may be dropped and **MCP** resumed with gradients calculated from the last primal feasible tableau. This is a promising direction to take, so long as the progress made by profit gradients is much better than that inherent in the cutting planes; if not, the additional burden of replicating the LP calculations each trial imply significant computational burden and perhaps practical failure for **MCP** since Gomory would clearly outperform it in such cases apart from round-off error.

Example 3: Haldi Non-convergence

In the non-convergent Haldi problem, which stalled in trial 30 in **Figure 26**, employment of the Gomory cut also enables convergence. The very first cut reduces φ^{30} to 57.27333. **MCP** is free to continue with Profit Bounds reduced by the new φ^{30}. However, trial 31 ends with another triplet of non-binary alternate optima and Gomory is employed again. In fact, in every subsequent trial, alternate triplets exist in the final tableau, requiring repeated use of Gomory to bypass each stall, resulting in the progress shown in **Figure 26**.

In the 38^{th} trial, **MCP** locates the solution: $X_1=X_2=X_6=0$, $X_3=X_4=X_5=X_7=X_8=X_9=X_{10}=1$, $\varphi^{38}=57$, which must be optimal. With tighter ε, additional trials are required ($\varepsilon=.001 \Rightarrow$ 125 trials, $\varepsilon=.0001 \Rightarrow$ 225 trials, etc.), but Gomory is not used after trial 37 in any case.

Though the use of the Gomory cut will be finite in theory, such an approach alone to extend **MCP** does not promise to be reasonably finite in practice; round-off error and

t	φ^t
31	57.27333
32	57.20297
33	57.18425
34	57.16051
35	57.15853
36	57.15720
37	57.15256
38	57.00000

Figure 26. Haldi Cut Progress

performance may very well render the algorithm quite useless in practical situations. Even when convergent, **MCP.0** does not appear to be efficient: Gomory solves the above problem in 13 iterations of dual simplex. In addition, even when such alternate optima

conditions do not exist and Gomory is not needed at all, the termination of the **MCP.0** algorithm under much more favorable conditions is still an open question due to the nature of the Epsilon neighborhood.

Overcoming Non-finiteness: the GCD

Now that the non-convergent nature of **MCP.0** is clearly demonstrated, and knowing that a procedure is technically available to overcome this condition, there yet remains the problem of the Epsilon neighborhood stopping criteria. As we observed in **Figure 11**, the published procedure for concluding the **MCP.0** algorithm is not robust in effect, presenting practical problems in both finiteness and accuracy. Defining extremely small bounds to guarantee convergence threatens to prolong the **MCP.0** convergence process and to introduce round-off error in such a manner as to make **MCP.0** inefficient. However, there is a convenient means to make it more efficient.

Consider another property of the LP search space that remains untouched heretofore: the nature of the profit values. When a strictly integer problem is being resolved (when all of the decision variables are integer variables), one thing is certain: the optimal profit function value will be some combination of the profits. If the current LP solution is not possibly some combination of these values, then the optimal profit function value has not been obtained. Furthermore, if any Profit Bound is not also some combination of these values, then the Profit Bound should be reduced to the largest such combination that is smaller than this bound.

Determining all possible combinations of the profits is itself a combinatorial problem, approaching the difficulty and inefficiency of solving the original problem itself by some Branch and Bound technique. However, there is a simple concept hidden here that does happen to be quite useful: the Greatest Common Divisor (**GCD**) of the Profit Function values. It is certain that the optimal Profit Function value will be a multiple of

the **GCD**.

Define **GCD** as the largest real number that will divide into all of the profit function costs with a zero remainder. Let a **GCD** function [ ] be defined as follows: $[\varphi]$ is the largest number $\leq \varphi$ that is a multiple of **GCD**. Now, enhance the definition of the Profit Bound as follows:

New Profit Bound $$\varphi_j^t = [\, \min(\varphi_j^{t-1},\ \varphi^t - (1 - X_j^t)\gamma_j^t) \,] \qquad \textbf{(eq. 4.1)}$$

This slight change in the definition of the Profit Bound actually resolves the problem of finiteness in a purely integer space. The Profit Bound is now forced to be a multiple of the **GCD**, which is itself a mere function of the decimal precision of the profit function values. This is evident from the fact that in this space any Profit Function value will be a sum of integer multiples of selected profits: the P_j. The **GCD** of the profits must therefore also be a divisor of the optimal solution to the problem. This implies that the number of **MCP.1** trials (Ω) required to determine the optimal profit function value is now upper-bounded by a linear function independent of problem size:

$$\Omega \leq [\varphi^t - \varphi] \,/\, \textbf{GCD} \qquad \textbf{(eq. 4.2)}$$

This is certainly a finite convergence criterion. Additionally, and conveniently, any minute change in any Profit Gradient γ_j^t for an optimal X_j=1 at any trial **t** where X_j^t<1 now implies a reduction in the corresponding Profit Bound of at least **GCD**:

$$\varphi_j^t \ \leq\ \varphi_j^{t-1} - \textbf{GCD} \qquad \textbf{(eq. 4.3)}$$

With a little experience in concluding **MCP.0** with the epsilon neighborhood, as in most algorithms, one begins to notice that the rate of progress toward the optimal solution diminishes over time as the algorithm converges: the objective function value moves rapidly at first, and then progress diminishes as the algorithm creeps closer and closer to the optimal solution. The fact that the **MCP** algorithm is gradient-based implies this type of convergence progress. Most of the progress toward the optimal solution is

made in the very early stages of the algorithm. Quite quickly in fact, the **MCP** gradient steps taken from one trial to the next become smaller than the **GCD** step size.

It is at this point that the **GCD** implies at least a ***linear*** convergence rate: the smallest step that can be taken between trials is the **GCD**. This conveniently eliminates a huge chunk of the "tail-end" of the convergence effort, the long and time-consuming process of whittling away at the search space with miniscule adjustments in the Profit Constraints to discover the optimal solution. Now, by definition, the step size is lower bounded by the **GCD**.

This simple development serves to completely eliminate the need for an epsilon neighborhood. The **GCD** has "discrete-ized" the solution space into a finite number of Ω inspection points: $\{\varphi^0, \varphi^1, \varphi^2, \ldots, \varphi^\Omega\}$, where $\varphi^n = [\varphi^n]$ when $n > 0$. Either the optimal solution lies at the current φ^t, which by nature of the newly defined Profit Bounds will also be a multiple of the **GCD**, or it does not. In either case, **MCP.1** or the cutting plane will either find the optimal solution in the current trial, or φ^t will be reduced by some integer multiple of **GCD** and the next point φ^{t+1} in the search space will be explored. The result: the epsilon neighborhood is no longer needed... at all. The **GCD** is certainly a practical step, developed with grade school arithmetic, and it has a significant affect upon the convergence properties of the algorithm.

Additional Efficiencies

There are a number of additional efficiencies that may be introduced into **MCP** to improve convergence performance. Most are derived from analysis of the Profit Gradients and Profit Bounds and are somewhat intuitive.

Gradient Test for Sub-Optimality

The first efficiency is rooted in the derivation and justification of the Profit Constraint (**eqs. 3.12, 3.13**). I have shown that positive Profit Gradients guarantee progress toward the optimal solution value when corresponding variables are unity in the optimal solution and non-integer in the current trial. Thus, under an **MCP.0** stall condition, when progress has not been made in the objective function value in the last trial, the gradients of all optimal solution variables that are not zero or unity in the current solution must be zero. What does this imply about non-integer-valued variables with positive gradients under a stall condition? Quite simply, such variables **must** not be members of the optimal solution: they are sub-optimal. If the objective function value does not decrease from one trial to the next any variable ending the previous trial with a finite, positive gradient and a fractional value (not zero or unity) may be permanently dropped from the search space as sub-optimal based upon this Gradient Test.

Min-Max Profit Bounds

Just as the Gradient Test implies efficiency during a stall condition, there are efficiencies rooted in the analysis of the Profit bounds in the context of the Profit Constraints. Suppose in an **MCP** problem at trial **t** there are a number (C) of Profit Constraints of X_j^t's, Y_j^t's, Z_j^t's,..., each having a number of active Profit Bounds: $\varphi_{X_1}^t,\ldots,\varphi_{X_a}^t$ in C_X, $\varphi_{Y_1}^t,\ldots,\varphi_{Y_b}^t$ in C_Y, $\varphi_{Z_1}^t,\ldots,\varphi_{Z_c}^t$ in C_Z, etc. In each Profit Constraint there is a maximum Profit Bound:

$$X_{max}^t = \max\left(\varphi_{X_1}^t, \varphi_{X_2}^t, \varphi_{X_3}^t,\ldots,\varphi_{X_a}^t\right)$$

$$Y_{max}^t = \max\left(\varphi_{Y_1}^t, \varphi_{Y_2}^t, \varphi_{Y_3}^t,\ldots,\varphi_{Y_b}^t\right)$$

$$Z_{max}^t = \max\left(\varphi_{Z_1}^t, \varphi_{Z_2}^t, \varphi_{Z_3}^t,\ldots,\varphi_{Z_c}^t\right)$$

...

etc., for all **C** binary sums. Let

$$\varphi^t_{mm} = \mathbf{min}\,(X^t_{max}, Y^t_{max}, Z^t_{max}, \ldots) \qquad \textbf{(eq. 4.4)}$$

Then it is true that:

$$\varphi \leq \varphi^t_{mm} \qquad \textbf{(eq. 4.5)}$$

The optimal solution value φ is upper bounded by the minimum from among all Profit Constraint maximal Profit Bounds. This follows from the fact that the largest Profit Bound in any binary sum is an upper bound on the profit function value φ since some member of each binary sum must be a member of the optimal solution. The smallest of all of these maximum Profit Bounds among all of the Profit Constraints is the most constraining of these maximum Profit Bounds.

Also, this minimum of the maximum Profit Bounds may be used to constrain all larger Profit Bounds to further tighten Profit Constraints. Any Profit Bound $\varphi^t_j > \varphi^t_{mm}$ should be reset to φ^t_{mm} before the Profit Constraints are formed for the next trial.

GLB: Greatest Lower Bound

Another efficiency introduced is the **GLB**: the ***Greatest Lower Bound***. This is the largest profit function value that has been determined from a feasible resolution at any point during solution development. It is basically a form of Branch and Bound and works on the same basic principle.

At the completion of each LP trial, and perhaps at strategic points during the trial (such as when the current profit function value is positive and the difference in value between tableaus is minimal), decision variables are intelligently forced to binary values and feasibility checked. If a feasible solution is found, then the profit function value is determined from this feasible solution and compared to the largest such value obtained thus far in the algorithm. If the new value is the largest thus far, then the potential

solution is retained along with the new objective function value, which then becomes the current **GLB**. If either the solution is infeasible, or the profit function value is less than the current **GLB**, then the result is discarded.

Intelligent forcing of variables to integer values is a bit more than rounding to the nearest integer, since it can be frequently expected in binary sums with many active variables that all of the values may be less than ½. It is understood that exactly one variable in each exclusion set will be unity, so rounding all variables is infeasible by definition in such cases. Setting the variable with the largest value in each exclusion set to unity is reasonable, implying that remaining variables in the exclusion set are set to zero.

The effect of employing the **GLB** concept is bi-directional movement toward the optimal solution. While **MCP.1** is restricting the solution space from above, the **GLB** is restricting the solution space from beneath. This has several beneficial properties.

First, by definition of **GLB**,

$$\varphi \geq \mathbf{GLB} \qquad \textbf{(eq. 4.6)}$$

By definition, the optimal solution is not less than any Greatest Lower Bound on the profit function. If, for some small positive ε, φ^t is found that is < **GLB** + **GCD** - ε, then the **GLB** solution is the optimal solution and the algorithm is complete. This follows since if φ^t is strictly < **GLB** + **GCD** and $\varphi \geq$ **GLB** and both **GLB** and φ are multiples of **GCD**, we must have φ = **GLB**.

Also, conveniently, if a **GLB** is determined at any point in the algorithm that is larger than some Profit Bound φ_j^t, I know that $X_j = 0$ in the optimal solution. This follows from the fact that the optimal profit function value φ is $\leq \varphi_j^t$ if $X_j = 1$ in the optimal solution and from the fact that $\varphi \geq$ **GLB**. Therefore, if φ_j^t is strictly < **GLB** I may deduce that $X_j = 0$ in the optimal solution and permanently eliminate X_j from the solution space. Any constraint in which such an X_j appears with a single other member X_k

implies $X_k = 1$ in the optimal solution. This development reduces problem size by eliminating variables (and therefore, at times, constraints) once their value in the optimal solution has been determined.

Further, and this will likely be a key differentiator for **MCP** in the field, any known profit function value can be supplied as the **GLB** in trial 1… even if the variable settings for this bound are unknown. Known bounds to the profit function can be used much as in any Branch and Bound technique to eliminate variables from the space with Profit Bounds less than this known **GLB**. Very large problems may reduce immediately to a more manageable size such that the optimal solution can be obtained more efficiently. If the optimal solution has been discovered to a large problem, **MCP.1** may be efficient in confirming this.

Limited Variable Employment

As in any branch and bound algorithm, efficiently removing variables and constraints from the problem space is quite beneficial in the resolution of large problems due to the fact that the singularly limiting factor in resolving such problems is *time*, which is directly related to problem size. It is well known that the relationship between resolution time and problem size is not generally linear in the number of decision variables (doubling the number of variables will likely MUCH more than double the resolution time). Any incremental improvement in problem size will therefore generally yield an exponential improvement in time to convergence; hence, developments in this area have a great deal of promise in extending the practical efficiency of the algorithm. The Profit Bound concept, by definition, suggests that it may become quite useful using such an approach.

In considering the Profit Bound further, it becomes apparent that if there were some way to determine an upper bound on the next LP solution, then variables might be

omitted from the next trial which cannot possibly be part of any optimal solution obtained in that trial. While in basic **MCP.0** there is no clear way to determine a reasonable upper limit on the objective function value that will be obtained in the next trial, this certainly is available as a benefit of the **GCD**.

It is true that, soon after beginning the **MCP.1** algorithm, relatively speaking, the **GCD** step size often becomes active in constraining the differences in the profit function value between successive trials, **t** and **t+1**; a linear convergence rate is generally determined in the system at this point. Once the **MCP.1** gradient-based progress is overtaken by the **GCD**, one may well predict the profit function values resulting from the **MCP.1** trials from this point on; the profit function values are clearly a linear function of the trial number **t**.

Consequently, it would appear that in any particular trial **t** in which the **GCD** is actively bounding the convergence rate, all Profit Gradients included in trial **t** should be equivalent to $\boldsymbol{\varphi}^t$: there is no benefit in including remaining variables in that trial. Disabling variables X_j with Profit Bounds $\boldsymbol{\varphi}_j^t$ less than the subsequent LP optimum $\boldsymbol{\varphi}^{t+1}$ will be termed "Limited Variable Employment" (**LVE**), and should be considered in resolving large problems.

Removal of variables due to Profit Bound reduction under **LVE** will result in:

1. obtaining the optimal solution,
2. the removal of enough variables to eliminate the potential of an optimal solution with remaining variables, or
3. a non-convergent stall condition.

If the optimal solution is precluded, the next upper bound in the convergence sequence should naturally be explored. In the non-convergent case, the use of a Gomory cut to discern the nature of the space should resolve the scenario cleanly.

To guarantee convergence within these last two techniques it is certainly

important that no variables be *permanently* discarded from the solution space so long as their Profit Bounds are not less than the **GLB**. Profit Bounds not less than the **GLB** must be archived for potential future use since there is no formal guarantee that such variables are indeed not part of the optimal solution. All that may be said with certainty is that such variables will be part of an *optimal* solution **iff** that solution value is less than or equal to their Profit Bound. If at any point in further solution development the next profit function value will not be greater than the Profit Bound of any temporarily excluded (disabled) variable, that variable must be reintroduced into the problem space before beginning the next trial. Basically, all variables with Profit Bounds greater than or equal to the next known Profit Function value at the end of any non-convergent trial must be included in the next trial in order to guarantee convergence.

Binary Sum Construction

In addition to the above efficiencies it is appropriate to consider how one might initially pose problems in a manner that leverages the strengths of **MCP**. Problem formulations may frequently be provided in a manner that is not suitable for **MCP** simply due to the fact that Binary Sum constraints are not commonly preferred by default. Certain types of constraints can as conveniently be represented by a set of Binary Sum constraints as not, allowing a tighter formulation for **MCP** rather than simply creating dummy variables and constructing the necessary Binary Sums (each of which imply a Profit Constraint) rather crudely, two variables at a time. Consider Example 1, my introductory example, given initially in **Figure 5**, reproduced below in **Figure 27**.

The second constraint C_2 has properties similar to a Binary Sum since it prevents choosing the pair X_1, X_3 and the pair X_1, X_4. Instead of adding four dummy variables and creating four additional Binary Sums as I did initially, I note that constraint C_2 implies the result in **Figure 28**.

max	X_1	X_2	X_3	X_4		
P_j	6	3	4	8		RHS
C_1	-2	0	4	2	<=	5
C_2	3	1	2	2	<=	4
C_3	1	1	1	1	<=	2

Figure 27. MCP.0 Overview Ex

C_{2A}	1	0	1	0	<=	1
C_{2B}	1	0	0	1	<=	1
C_{2C}	0	1	1	1	<=	2

Figure 28. Implied Constraints

The fact that these three constraints also imply C_2 allows me to employ C_{2A} and C_{2B} and use two proxy variables to compose two Binary Sums instead of three, one for each of X_1, X_3 and X_4. I may also observe that constraint C_{2C} is redundant since it is implied by C_3. This insight implies the following formulation:

max	X_1	X_2	X_3	X_4	X_5	X_6	X_7	
P_j	6	3	4	8	0	0	0	RHS
C_1	-2	0	4	2	0	0	0	5
C_2	1	1	1	1	0	0	0	2
C_3	1	0	1	0	1	0	0	1
C_4	1	0	0	1	0	1	0	1
C_5	0	1	0	0	0	0	1	1

Figure 29. Reformulated Overview Ex

Rather than the seven constraints and eight variables in the original **MCP** formulation, I

now have five constraints and seven variables. I also have Binary Sums that are more significantly constraining than those in the original formulation. Due to this increased tightness in the Binary Sum constraints it so happens that LP by itself resolves this formulation optimally in trial 0... **MCP** is never even employed!

Certainly, choosing to conveniently represent basic constraints as Binary Sums will favor **MCP**. One should examine any formulation supplied to **MCP** to note such opportunities. However, creating an algorithm to search for such opportunities and leverage them is a bit more challenging.

One very simple heuristic suggested by the above example is to create a Binary Sum constraint from any homogeneous constraint (no negative coefficients) that has at least two large technical coefficients with respect to the right hand side, such that every variable with ratio $b_i / a_{ij} < 2$ in the noted constraint is a member of the Binary Sum.

A second similar heuristic is to create a Binary Sum from each set of variables that are mutually exclusive due to the size of their technical coefficients (which implies constraints $\boldsymbol{C}_{2A}$ and $\boldsymbol{C}_{2B}$ from $\boldsymbol{C}_2$ above). This second heuristic by itself enables LP to resolve my first example problem, as in my more comprehensive re-formulation but with slightly more effort, since retaining the original constraint $\boldsymbol{C}_2$ is non-binding on the LP solution and merely adds a few more pivots in the simplex process.

Binary Expansion

What has been sacrificed in developing these efficiency extensions to the basic **MCP.0** algorithm is perhaps not apparent at first; a significant restriction has been added to the constraints: all variables must be *binary*. In most non-binary scenarios, integer variables and rational variables with finite precision can be expanded into a set of binary numbers.

Take any number $\boldsymbol{X}_j$ (Integer, or Rational with finite decimal precision) and find a

Least Upper Bound for it from the LP constraints. In any constraint **i** containing X_j that has all coefficients of the same sign as the (non-zero) right side b_i, find the ratio b_i/a_{ij} (also use non-zero P_j/φ^0 if all of the profit function values P_j are the same sign as initial LP solution φ^0). The minimum of these ratios serves as U_j, the Least Upper Bound to X_j.

Determine the smallest integer E such that $2E > U_j$. Replace U_j with U_j - 2E-1 and create a binary variable X_j with profit function value $P_j = 2P_j$E-1 and $a_{ij} = 2a_{ij}$E-1 values in all of the constraints. Continue this process by finding another E such that 2E > the new U_j until the difference U_j - 2E-1 is ≤ 1, implying that X_j itself will carry the first binary digit in its binary expansion. Then, for non-integer X_j, determine E such that 2–E is smaller than the smallest decimal precision required of X_j in the solution. Form E binary variables with factors 2^{-1}, 2^{-2}, 2^{-3},..., 2^{-E} on the P_j and a_{ij}. All resulting variables form a binary representation for X_j.

A mapping of all binary expansion variables and their exponents to the respective "parent" X_j must be retained to enable reconstruction of the value of the original X_j in the optimal solution. The correct value for X_j can be determined at the termination of **MCP.1** by summing the products of the values of all the variables in the binary representation of X_j and their corresponding factors.

Detailed Outline

A good bit of algorithmic complexity has been introduced in extending the basic **MCP.0** algorithm. A thorough outline of the steps, as well as a flowchart of the solution structure, is presented in a summary of the entire algorithm. To simplify the outline, a sub-algorithm that occurs multiple times is *italicized*, underlined, and only defined the first time it appears in the outline (e.g. ***Find new GLB***)

0. Set up

0.0 Load problem, maximizing use of Binary Sums to represent constraints

0.1 Calculate **GCD**

0.2 If the problem is a minimization problem, convert to maximization

0.3 Check constraint validity

0.4 Set **GLB** (Greatest Lower Bound on optimal solution) = -infinity

0.5 Set **t** = -1

0.6 If non-binary variables exist,

0.6.1 run the LP Simplex Method to completion for trial **t** = -1

0.6.2 Put problem in pure binary form

0.7 **t = t + 1**

0.6 Run the LP Simplex Method to completion for trial **t** = 0.

0.7 Set $\boldsymbol{\varphi}^0$ to the **GCD** reduction of the initial LP solution, [$\boldsymbol{\varphi}^0$], in trial **t** = 0.

0.8 Calculate Gradients / Profit Bounds and set up Profit Constraints.

1. Run MCP.1 trial t

1.0 Revise Constraints.

1.0.1 *Reduce Problem Space as needed.*

1.0.1.1 Set any active X_j alone in an equality constraint to b_i / a_{ij} and remove both.

1.0.1.2 Set X_j alone in inequality constraint **i** and not active in any other constraint with a positive profit to max (1, [b_i / a_{ij}]) and remove both.

1.0.1.3 If $b_i = X_j = 1$ deactivate remaining variables in exclusion set of X_j.

1.0.2 Activate all variables needed for trial run.

1.0.3 Update Profit Constraint coefficients with new Profit Bounds.

1.1 Solve the LP to get a new LP solution $\varphi^t = [\varphi^0]$

1.2 *Find new* ***GLB***

1.2.1 For each Exclusion Set force the largest X_j to 1, remaining variable(s) to 0.

1.2.2 If resulting solution φ^{GLB} is feasible, set **GLB** = max(**GLB**, φ^{GLB}).

1.2.3 If **GLB** = φ^t report current solution φ^{GLB} as optimal and stop.

2. Calculate Profit Bounds: φ_j^t

2.1 Calculate Profit Gradient γ_j^t for all X_j from tableau state **t** for **MCP.1** trial **t**+1.

2.2 For all active X_j calculate Profit Bound φ_j^t, and find the Min/Max bound

2.4 Set φ_j^t = min(Min/Max, $[\varphi_j^t]$)

2.5 Permanently disable any variable X_j with $\varphi_j^t <$ **GLB**.

2.6 (LVE) Temporarily disable any variable X_j with $\varphi_j^t < \varphi^t$ - **GCD**

3. If there is no improvement in the Profit Function a stall point has been reached.

3.1 Permanently remove all variables with positive finite gradients.

3.2 If active Profit Bounds are not all equivalent, employ a local **LVE**.

3.2.1 Temporarily deactivate any X_j with $\varphi_j^t < \varphi^t$.

3.2.2 If current objective function value cannot be reached with remaining X_j then.

3.2.2.1 Set $\varphi^{t+1} = \varphi^t - \mathbf{GCD}$.

3.2.2.2 If $\varphi^{t+1} = \mathbf{GLB}$, report optimum & stop.

3.2.2.3 For active X_j set φ^{t+1} to $\min(\varphi_j^t, \varphi^{t+1})$

3.2.2.4 $\mathbf{t} = \mathbf{t} + 1$

3.2.2.5 Run **MCP.1** trial t (go to 1.0)

3.2.3 Run Simplex. If resulting φ^t is reduced from pre-LVE φ^t,

3.2.3.1 $\mathbf{t} = \mathbf{t} + 1$

3.2.3.2 Run **MCP.1** trial t (go to 1.0)

3.3 Perform **GOMORY**:

3.3.1 Construct a finite Gomory cut for each non-integer valued variable.

3.3.2 Dual simplex until either:

3.3.2.1 the LP solution in any Dual Simplex tableau is $< \varphi^t$, implying the stall point has been "jumped" and $\varphi < \varphi^t$.

3.3.2.1.1 Set $\varphi^{t+1} = \varphi^t - \mathbf{GCD}$

3.3.2.1.2 If $\varphi^{t+1} = \mathbf{GLB}$, report optimum & stop.

3.3.2.1.3 For active X_j set φ^t to $\min(\varphi^t, \varphi^{t+1})$

3.3.2.1.4 $\mathbf{t} = \mathbf{t} + 1$

3.3.2.2 The LP solution is primal feasible with the solution $\varphi^* = \varphi^t$

3.3.2.2.1 *Get new GLB*. If $\mathbf{GLB} = \varphi^t$, report optimum & stop.

3.3.2.2.2 Get Profit Bounds

3.3.2.2.3 If φ_j^t are not equivalent, disable all X_j with $\varphi_j^t < \varphi^t$

3.3.2.2.4 Repeat **GOMORY** (go to 3.3)

4. Run **MCP.1** trial t (go to 1.0)

Flowchart

The flow chart below completes the description of the enhanced **MCP.1** algorithm.

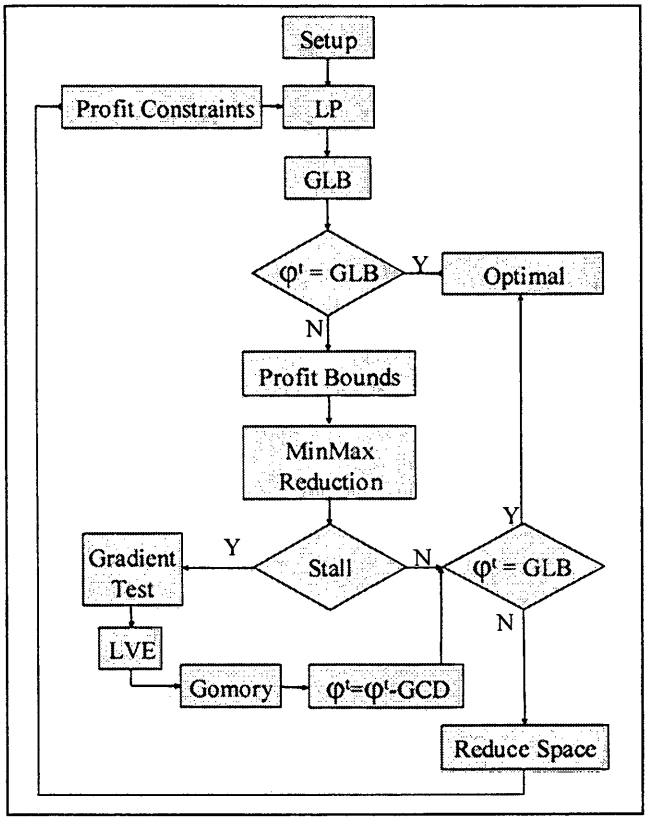

Figure 30. MCP.1 Flowchart

I have now presented all of the relevant theoretical detail concerning **MCP**, from defining both its mathematical and practical contexts to a full elaboration of how it works. Motivated by an expectation that the gradient-based nature of **MCP** holds much practical value I have exposed and explored its inefficiencies, showing that **MCP.0** is neither finite nor convergent. I have also proposed enhancements which theoretically overcome its limitations and I have presented a complete workflow that includes all of these enhancements. I have yet to show how the enhanced algorithm performs in benchmark testing, and will conclude with a discussion of further research areas that should be pursued.

CHAPTER 5

RESULTS

With both the nature and context of **MCP.0** fully developed, having shown that it is neither finite nor convergent, and having proposed a number of enhancements that theoretically overcome its limitations, I present results from initial benchmark testing of the enhanced **MCP.1**.

Additional constraints have been imposed and a great deal of complexity has been imbedded in the extended **MCP.1** algorithm. The solution space has been restricted to 0-1 programming, requiring a significant increase in problem size to return to problems in MILP. Also, **MCP.1** performs better with integer values as opposed to small fractional components in the profit function. This increase in complexity may be offset by an increase in performance.

What has been gained in performance? A finite step size is realized, implying a linear progression towards optimality; the epsilon neighborhood difficulty has vanished; the elegance of the Profit Constraint is presented in practice, and problem size is potentially significantly reduced as the algorithm proceeds. These tradeoffs have been evaluated in a number of practical applications.

Example 1 Revisited

Reconsider the problem given in Example 1, **Figure 5**, my introductory illustration.

max	X_1	X_2	X_3	X_4		
P_j	6	3	4	8		RHS
C_1	-2	0	4	2	<=	5
C_2	3	1	2	2	<=	4
C_3	1	1	1	1	<=	2

Figure 31. MCP.1 Intro Ex

Clearly, there was difficulty in properly concluding the algorithm: an optimal result was never obtained with **MCP.0**. In addition to my earlier observation that a clever analysis of the problem constraints permits LP to find the optimal solution without employing **MCP** at all, what do remaining proposed efficiencies imply if we retain the original formulation?

First, a **GCD** of 1 is calculated from the profit function values. This implies that the initial Profit Bound of 12.5 for X_4 and X_7 in trial 0 may be reduced to 12 as follows:

t = 1	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
X_j^0	$\frac{1}{2}$	$\frac{1}{2}$	0	1	$\frac{1}{2}$	$\frac{1}{2}$	1	0
γ_j^0	3	3	$\frac{1}{2}$	M	1	1	M	3 ½
φ_j^0	11	11	12	12 ½	12	12	12 ½	9
$[\varphi_j^0]$	11	11	12	12	12	12	12	9

Figure 32. MCP.1 Introductory Ex: Trial 0

After forming the Profit Constraints for trial 1, **MCP.1** concludes trial 1 with an LP solution of 11 $^{11}/_{12}$ (as in **MCP.0**). Results of **MCP.1** calculations for trial 1 are:

t = 1	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
X_j^1	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{19}{24}$	1	$\frac{11}{12}$	$\frac{11}{12}$	$\frac{5}{24}$	0
γ_j^1	1	1	$\frac{2}{3}$	M	2	$\frac{1}{3}$	$\frac{2}{5}$	$\frac{1}{2}$
φ_j^1	11	11	$11\frac{7}{9}$	$11\frac{11}{12}$	$11\frac{3}{4}$	$11\frac{8}{9}$	$11\frac{3}{5}$	$11\frac{5}{12}$
$[\varphi_j^1]$	11	11	11	11	11	11	11	11
best	11	11	11	11	11	11	11	9

Figure 33. MCP.1 Introductory Example: Trial 1

Trial 2 concludes with an objective function value of 11 and Profit Bounds unchanged from trial 1 except φ_3^2, which is now 10. The only positive Profit Gradients are now γ_3^2=1.33 and γ_8^2=2, **Figure 34**.

t = 2	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
X_j^2	0	0	$\frac{3}{4}$	1	1	1	$\frac{1}{4}$	0
γ_j^2	0	0	$\frac{4}{3}$	0	0	0	0	2
φ_j^2	11	11	$10\frac{2}{3}$	11	11	11	11	9
$[\varphi_j^2]$	11	11	10	11	11	11	11	9
best	11	11	10	11	11	11	11	9

Figure 34. MCP.1 Introductory Example: Trial 2

Trial 3 concludes with the same objective function value of 11. The **Gradient Test** therefore implies that X_3 and X_8, both less than unity with positive gradients in trial

2, are sub-optimal. Their binary sum counterparts X_7 and X_4 are therefore optimal. Only half the variables remain: X_1, X_2, X_5 and X_6.

t = 3	X_1	X_2	X_5	X_6
X_j^3	$\frac{1}{2}$	0	$\frac{1}{2}$	1
γ_j^3	0	0	0	M
φ_j^3	11	11	11	11
$[\varphi_j^3]$	11	11	11	11
best	11	11	11	11

Figure 35. MCP.1 Introductory Example: Trial 3

Analysis of the problem constraints is as appropriate after a reduction in the problem size as it is during initial formulation. The updated formulation is given in **Figure 36**.

Note that X_1 is sub-optimal since constraint C_2 requires $X_1 = 0$. This implies that X_2 is optimal, that the optimal solution is $X_2 = X_4 = X_5 = X_7 = 1$ and that $\varphi = 11$. Voila!

max	X_1	X_2		
P_j	6	3		RHS
C_1	-2	0	<=	3
C_2	3	1	<=	2
C_3	1	1	<=	1

Figure 36. MCP.1 Updated Problem

Compare the directness of **MCP.1** with **MCP.0** and Gomory:

Trial	MCP.1	MCP.0	GOMORY
1	11.92	11.916667	11.800
2	11.00	11.731302	11.778
3	11.00	11.587084	11.750
4		11.401768	11.750
5		11.226429	11.667
6		11.136986	11.667
7		11.076570	11.500
8		11.043031	11.500
9		11.021360	11.000
10		11.009764	
11		11.003431	
12		11.000925	
13		11.000136	
14		11.000011	
15		11.00000022	
16		11.0000000013	
17		11.00000000000013	
18		Failure	

Figure 37. Example 1: Enhanced Results

Balas converges in iteration 23. The improvement is perhaps enough comment.

Example 3 Revisited

Reconsider the non-convergent problem of Haldi.

max	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	
P_j	20	18	17	15	15	10	5	3	1	1	**RHS**
C	30	25	20	18	17	11	5	2	1	1	≤ 65

Figure 38. Haldi: Non-Convergent Example

MCP.0 failed to converge. The use of Gomory with **MCP.0** yields an optimal solution after 47 total trials, while Gomory by itself converges in 13 complete dual simplex iterations. What do additional efficiencies of the **GCD**, the **Gradient Test**, **LVE**, **Min-Max Profit Bounds** and **Profit Constraint Retirement** imply?

t	MCP.0 φ^t	t	MCP.0 φ^t	t	Gomory φ^t	t	MCP.1 φ^t
0	58.66667	16	57.85437	0	58.667	0	58
1	58.50000	17	57.84774	1	58.588	1	57
2	58.35135	18	57.84264	2	58.500	2	57
3	58.33286	19	57.83890	3	58.067		
4	58.25354	20	57.83635	4	57.923		
5	58.16587	21	57.83477	5	57.857		
6	58.09908	22	57.83393	6	57.833		
7	58.04647	23	57.83355	7	57.800		
8	58.00446	24	57.83341	8	57.750		
9	57.97049	25	57.83336	9	57.667		
10	57.94276	26	57.83334	10	57.500		
11	57.92000	27	57.83334	11	57.000		
12	57.90126	28	57.83333	12	57.000		
13	57.88581	29	57.83333	13	57.000		
14	57.87311	30	57.83333				
15	57.86274		**Failure**				

Figure 39. Haldi: Enhanced Results

At the end of trial 0 in **MCP.1**, rounding each variable to the nearest integer yields a **GLB** of 52 and the **GCD** of 1.0 implies a reduction of φ^* from 58.7 to 58. All Profit Bounds are bounded by the reduced φ^* and trial 1 is begun. At the end of trial 1 it comes as no surprise that alternate optima are discovered and **MCP.1** is stalled. However, under **LVE** a number of variables can be temporarily removed from the problem space since an optimal solution of $\varphi^* = 58$ could not contain any of them ($\varphi_1^1 = 53$, $\varphi_2^1 = 55$,

$\varphi_{14}^1 = \varphi_{15}^1 = \varphi_{16}^1 = \varphi_{17}^1 = \varphi_{18}^1 = 57$). Their respective binary sum partners are temporarily set to unity and the corresponding binary sum constraints are removed. The remaining variables all have a Profit Bound of 58. The resulting problem in six variables and four constraints (from 20 variables and 21 constraints) again concludes with φ^* at 58. However, recalculating the Profit Bounds reveals that in this limited set of variables proxy variable X_{13} has a Profit Bound of 50 so it can be temporarily removed and X_3 temporarily set to unity. This implies $X_3 = X_4 = X_5 = X_7 = X_8 = 1$, yielding a $\varphi^* = 65$ and inconsistent with my current φ^*. It can thus be concluded that the optimal solution is not 58 and that $\varphi^* \leq 57$. All disabled variables are restored and the algorithm continues with all Profit Bounds bounded by 57.

Again, as expected, trial 2 concludes with a stall point for **MCP.0**. Profit Bounds are similar to trial 1, yet since $\varphi^* = 57$ only X_1 and X_2 are removed (along with their proxy counterparts X_{11} and X_{12}). The trial is rerun on the reduced space and the φ^* value of 57 is reached again in the final simplex tableau. All Profit Bounds remain at 57 so Gomory is employed to bypass the stall point or locate the optimal solution. After eleven distinct dual simplex iterations the optimal solution $\varphi^* = 57$ is obtained.

MCP.1 thus converges on a non-convergent **MCP.0** problem. The convergence is enabled by enhancements to the basic **MCP.1** algorithm. However, Gomory appears more efficient on this problem since very little actual solution progress was made by **MCP** itself. The total number of pivots needed by Gomory is 48 while **MCP.1** requires 108, which, though the problem size is reduced in **MCP.1**, is likely a more accurate measure of performance than the number of trials since a Gomory trial is not the same as an **MCP.1** trial by definition. This poorer performance is likely related to the fact that the original formulation of this problem has no inherent multiple choice constraints and is replete with alternate optima by design.

Example 4: Dam Siting in the Delaware River Basin

The most famous example of modeling water facility placement is likely the dam siting model of Peter O. Steiner [17]. Woolsey [24] observes that two types of IP algorithms were used to resolve 352 instances provided by Maynard Hufschmidt (of the U.S. Bureau of Reclamation). Since Steiner's model is replete with binary sums it appears well suited for **MCP**. I present formulation detail before employing **MCP.1** to resolve a particular instance of the model provided by Dr. Hufschmidt.

Let binary X_{ij} represent ***build*** or ***do not build*** a dam at site **j** for purpose **i** in the Delaware River Basin (e.g. at each of three distinct sites **j** in Hawk Mountain I model three types of dams fulfilling purposes **i**: flood control, hydro-power, and combined multi-purpose use). Let P_{ij} be the public benefit and C_{ij} the construction cost of a dam at site **j** with purpose **i**. Maximize public benefit P in constructing dams at nine potential sites situated within four regions (Hawk Mountain, Tock Island, Bear Creek and Maiden Creek), with at most one dam per region, subject to B: the public sector budget for dam construction. Let private sector alternatives have only one purpose (hydro-power, requiring only site subscript **j**).

The objective function is:

$$P = \sum_{i=1}^{i=3}\sum_{j=1}^{j=3} P_{ij}X_{ij} + \sum_{i=4}^{i=7}\sum_{j=1}^{j=3} P_{ij}X_{ij} + \sum_{j=2}^{j=3} P_jX_j + \sum_{i=8}^{i=10}\sum_{j=4}^{j=6} P_{ij}X_{ij} + \sum_{j=5}^{j=6} P_jX_j + \sum_{i=11}^{i=13}\sum_{j=7}^{j=9} P_{ij}X_{ij} + \sum_{j=8}^{j=9} P_jX_j$$

Hawk Mtn. + Tock Island + Alt + Bear Creek + Alt + Maiden Creek + Alt

The budget constraint is: $\sum_{i=1}^{i=7}\sum_{j=1}^{j=3} C_{ij}X_{ij} + \sum_{i=8}^{i=10}\sum_{j=4}^{j=6} C_{ij}X_{ij} + \sum_{i=11}^{i=13}\sum_{j=7}^{j=9} C_{ij}X_{ij} \le B$.

In each region at most one dam one may be built:

Hawk Mtn. $\sum_{i=1}^{i=3}\sum_{j=1}^{j=3} X_{ij} + X_2 + X_3 \le 1$ Tock Island $\sum_{i=4}^{i=7}\sum_{j=1}^{j=3} X_{ij} + X_2 + X_3 \le 1$,

Bear Creek $\sum_{i=8}^{i=10}\sum_{j=4}^{j=6} X_{ij} + X_5 + X_6 \le 1$ Maiden Creek $\sum_{i=11}^{i=13}\sum_{j=7}^{j=9} X_{ij} + X_8 + X_9 \le 1$.

In the published formulation [24] an additional twelve constraints are included to accommodate incompatibility of purpose at each site. However, I note that these purpose constraints are redundant with the four region constraints given above, which allow at most one dam per site, and therefore omit them.

A complete formulation with an actual budget, profits and costs is given below:

max	X11	X12	X13	X21	X22	X23	X31	X32	X33	X41	X42	X43	X51	X52	X53	X61	X62	X63	X71	X72	X73	X84	X85	X86	X94	X95	X96	X104	X105	X106	X117	X118	X119	X127	X128	X129	X137	X138	X139	X2	X3	X5	X6	X8	X9	
Pij	82	138	147	116	190	298	140	232	239	251	315	343	278	360	387	315	395	431	331	418	443	58	103	122	104	147	155	130	200	217	125	152	161	160	205	213	186	269	340	70	58	50	44	70	66	RHS
R1	160	220	230	240	300	320	300	380	410	490	550	570	520	600	630	550	650	690	590	720	770	120	170	180	160	230	250	200	300	330	150	180	190	200	250	270	240	310	340	0	0	0	0	0	0	<= 1300
R2	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	<= 1
R3	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	<= 1
R4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	<= 1
R5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	<= 1

Figure 40. DRB: Dam Siting

Variable names are given in the first row, followed by the profit function values P_{ij}. In the next row there is the single budget (resource) constraint, followed by the four region constraints. **GCD** is unity. The finaltableau of trial **0** yields an initial upper bound to the optimal solution of $\varphi^0 = 1087$. Four Profit Constraints are then formed corresponding to the four binary sums.

During trial 1, after only 27 total pivots, **MCP.1** locates the **GLB** of 1075 below:

X_{ij}	P_{ij}
X_{23}	298
X_{53}	387
X_{139}	340
X_5	50

Figure 41. DRB: GLB

Trial 1 concludes with $\varphi^1 = 1080$. Of the 45 Profit Bounds from trial 0, only seven satisfy this **GLB**:

X_{ij}	φ^0_{ij}	X_{ij}	φ^0_{ij}
X_{23}	1080	X_{139}	1080
X_{53}	1080	X_5	1080
X_{62}	1076	X_6	1080
X_{63}	1079		

Figure 42. DRB: Persisting variables

All other variables are eliminated; their Profit Bounds are strictly less than this **GLB** so the **Gradient Test For Sub-Optimality** implies they are sub-optimal. Further, X_{23} and X_{139} are the only remaining variables in two binary sums so they are optimal. These two constraints and their corresponding profit constraints are eliminated. Trial 2 begins with the problem reduced to 5 variables and 5 constraints as in **Figure 47**. Trial 2 concludes with $\varphi^2 = 1079$ and trial 3 concludes with $\varphi^3 = 1075$, indicating that the **GLB** is optimal.

The foregoing example illustrates the elegance of the extended gradient-based **MCP.1** approach in solving a practical problem.

Max		X_{53}	X_{62}	X_{63}	X_5	X_6	
	P_{ij}	387	395	431	50	44	**RHS**
S_{46}	0	63	65	69	0	0	64
S_{48}	-M	1	1	1	0	0	1
S_{49}	-M	0	0	0	1	1	1
S_{50}	0	-54	-43	-10	50	44	0
S_{51}	0	387	395	431	-392	-398	0
	RC	M	M	M	M	M	-M

Figure 43. DRB: GLB variables

Compare results from **MCP.0** (epsilon = .01) in **Figure 44**.

	MCP.0		MCP.0		MCP.0		MCP.0
t	φ^t	t	φ^t	t	φ^t	t	φ^t
1	1081.259	8	1077.175	15	1076.057	22	1075.531
2	1080.083	9	1076.942	16	1075.938	23	1075.461
3	1079.164	10	1076.666	17	1075.900	24	1075.443
4	1078.838	11	1076.536	18	1075.837	25	1075.000
5	1078.116	12	1076.427	19	1075.719		
6	1077.832	13	1076.317	20	1075.661		
7	1077.576	14	1076.133	21	1075.571		

	MCP.1
t	φ^t
1	1087
2	1079
3	1075

Figure 44. DRB: Results

With enhanced **MCP.1** Gomory is never employed, even when **GCD** is not invoked. Though this problem is small the efficiency of enhanced **MCP** is notable. Pure Gomory requires 39 pivots with the full problem scope, in four complete dual simplex iterations. The enumerative technique of Balas takes well over one thousand iterations to converge. Clearly, the extensions to **MCP.0** are efficient in this case.

Consider variations of this problem assuming different Budgets. **Figure 49** gives the respective pivots needed by Gomory and **MCP.1**.

B	GOM	MCP.1
1600	FAIL	156
1500	FAIL	401
1400	17	46
1200	131	1320
1100	FAIL	2802
1000	FAIL	193
900	FAIL	727
800	FAIL	1473

Figure 45. DRB: Variations

MCP.1 evidences significant efficiency in comparison to Gomory, which fails in six of the eight trials, performing well only in the neighborhood of the original budget. The robustness of **MCP.1** is evident in this test. Fundamentally, it is the nature of the Profit Gradient at work.

MIPLIB

Consider preliminary results from MIPLIB, the benchmark test suite from Rice University (Rice 1996). These problems are unusually difficult, largely degenerate and academic (among small binary tests, only problem mod008 has known application). Enhanced **MCP** performs well on twelve smaller knapsack-type problems, solving four of them optimally. Pure Gomory fails due to round-off error in each case.

In other smaller MIPLIB problems round-off error compounds to hinder

NAME	Constraints x Variables	PIVOTS
lseu	28 x 89	21096
mod008	6 x 319	455
p0033	16 x 33	5130
stein27	118 x 27	14681

Figure 46. MIPLIB Results

resolution. The current implementation of **MCP.1**, including code for Gomory cuts, does not yet handle round-off error with industrial rigor. Further work in this area may yield additional optimal results in MIPLIB. Also, 64-bit architecture employed for precision limits testing **MCP** on larger problems due to current memory constraints in the test environment (1 GB).

Knapsack Packing

Fifty-four knapsack problems collected by Carnegie Mellon University [5] are optimized with **MCP.1**. "Objects to pack" range from ten to ninety, and knapsacks range from four to thirty. Enhanced **MCP.1** solves 100% of these problems optimally, as in **Figure 51**.

NAME	PIVOTS	NAME	PIVOTS	NAME	PIVOTS
FLEISHER	38	WEI SHIH 03	2768	WEI SHIH 21	340
HANSEN, PLATEAU 1	274	**WEI SHIH 04**	147	WEI SHIH 22	2,586
HANSEN, PLATEAU 2	77,124	WEI SHIH 05	53	WEI SHIH 23	29,071
PB 1	538	WEI SHIH 06	225	WEI SHIH 24	813
PB 2	32,203	WEI SHIH 07	1784	WEI SHIH 25	3,076
PB 4	505,670	WEI SHIH 08	474	WEI SHIH 26	1,877
PB 6	47,630	**WEI SHIH 09**	48	WEI SHIH 27	1,309
PB 7	35,377	WEI SHIH 10	2001	WEI SHIH 28	111
PETERSEN 2	558	WEI SHIH 11	59	WEI SHIH 29	318
PETERSEN 3	118	WEI SHIH 12	1820	**WEI SHIH 30**	1,077
PETERSEN 4	29	WEI SHIH 13	172	WEINGARTNER 1	256
PETERSEN 5	3,917	WEI SHIH 14	128	WEINGARTNER 2	366,728
PETERSEN 6	1,193	WEI SHIH 15	1608	WEINGARTNER 3	129,537
PETERSEN 7	140,924	WEI SHIH 16	1739	WEINGARTNER 4	428
SENJU, TOYODA 1	33,314	WEI SHIH 17	208	**WEINGARTNER 5**	441
SENJU, TOYODA 2	20,685	WEI SHIH 18	2841	WEINGARTNER 6	24,758
WEI SHIH 01	265	WEI SHIH 19	691	WEINGARTNER 7	3,199
WEI SHIH 02	1031	WEI SHIH 20	1029	WEINGARTNER 8	264,505

Figure 47. Carnegie Mellon Knapsack Packing Results

Resolution time is up to three minutes (for three outliers), but is generally a matter of a few seconds (600 MHz UNIX Alpha processor). Gomory resolves only four of these test problems (**bold**), in all other cases failing due to round-off error.

These testing results are certainly preliminary, but they do appear to indicate that **MCP.1** performs well, particularly on multiple choice problems. It has performed quite well on a famous practical problem in site allocation that is characterized by binary sums. Given the theoretical strengths of **MCP** this appears to be a consistent result.

CHAPTER 6

FURTHER RESEARCH

If initial benchmark testing is any indication, **MCP.1** does appear to be potentially quite useful in practice. The general importance of the "multiple choice" decision scenario and **MCP**'s prominent place within this context, especially now that I have enhanced it to be both convergent and finite, suggests that any further effort expended in developing the efficiency of this algorithm has the potential to be valuable.

LVE

The **Limited Variable Employment** enhancement can be employed during a stall condition prior to invoking Gomory, as well as when convergence progress is bounded by **GCD**. In this case the lower bound on the objective function is the current lower bound φ^t rather than φ^t - GCD. This step requires re-running the LP on the reduced model prior to employing Gomory, with the advantage of a reduced problem size until the stall point is overcome.

It is unclear whether such use of LVE will be effective in larger problems. Tests on various types of live problems should be performed accordingly to determine if the additional computational burden is compensated by faster progress toward optimality. **MCP** can certainly function well without **LVE**, and it is difficult to tell with small test problems whether this enhancement is practical.

Revised Simplex Updates

Much of the limitation of **MCP.1** lies in having to completely rerun the LP to update the Profit Constraints for each new trial. When the progress of the algorithm does not happen to reduce the problem space dramatically through the efficiencies implied by the Profit Gradient, such a computational burden tends to offset the strength inherent in **MCP**'s ability to modify the search space without adding new constraints.

To improve performance, standard tricks are available in the revised Simplex method that permit updating the simplex tableau with revised Profit Constraints without needing to re-run the entire LP from scratch. These approaches may significantly reduce the computational burden in many situations, resulting in an overall improvement for **MCP.1**.

Special care will need to be taken when considering the affect of such maneuvers on the sundry enhancements suggested to the algorithm. Temporarily dropping variables and redundant constraints, reintroducing variables or constraints or noting when such are permanently obsolete, accommodating changes in **GCD** bounds and **GLB** bounds; all of these areas must be carefully reconsidered in light of the added complexities implied in updating the simplex tableau. It is likely, under conditions implying considerable model size reduction, that starting the LP afresh will be more efficient than updating the tableau by other means. Such conditions and their thresholds will need to be validated mathematically and explored with extensive testing. In my opinion, this approach is very promising and should definitely be the next area of focus in **MCP** research.

GCD Cuts

The tendency of **MCP** to stall frequently has been observed on many difficult benchmark problems. Most of the problems in the MIPLIB suite exhibit this behavior with **MCP**. A significant number of large "perfect" **MCP** problems were also generated

in my research to better predict whether this behavior will be consistent with **MCP**. Such "perfect" problems were constructed using random number generators such that every decision variable was both binary and a member of a binary sum, and every profit an integer: the ideal scenario for **MCP**. Every single trial after the first (very) few trials in such problems required the employment of GCD cuts and Dual Simplex, with step sizes between iterations being limited to **GCD**. Convergence was thus relatively slow compared to what it would be if the profit constraints remained effective, implying a complete LP iteration for most every unit of distance between the initial **MCP** solution φ^0 and the optimal solution φ. Improvements in **MCP** using revised simplex updates may improve the performance of **MCP** considerably here.

MCP.1 may perform better than some conventional algorithms, particularly if the **GCD** is large, but programs such as CPLEX appear to outperform **MCP.1** in smaller problems when the **GCD** is unity, implying that **MCP.1** may not generally be more efficient on some varieties of practical problems than existing algorithms. Based on my limited computational experience, unless the progress made by the Profit Constraints exceeds that of the **GCD**, **MCP** certainly appears to be a less efficient way to resolve such problems. This is due to the fact that **MCP** itself is adding no value in the solution progress while it is creating significant computational burden: all the progress is achieved by Gomory, **GLB**, and **GCD**.

The use of these three algorithms, Gomory, **GLB** and **GCD**, by themselves, appears to promise significant value in such environments beyond the scope of **MCP**. The benefit in eliminating **MCP** might be found in being able to construct a "large" **GCD** cut based on the information provided by the **GCD** without having to rerun the LP to apply the constraint. Such an approach would involve maintaining an inverse matrix to aid in constructing the cut. This area should definitely be explored in addition to the above research directly involving **MCP**.

MCP certainly appears to hold a prominent place in a very practical context. I

have presented its nature has in great detail, exposing its limitations and strengths, and I have enhanced it to be both convergent and finite. Its testing results are initially positive, indicating that further research with MCP is valuable to pursue.

CHAPTER 7

CONCLUSION

This current work re-introduces Multiple Choice Programming, **MCP**, the unique binary linear optimization heuristic introduced by W. C. Healy, after a long period of relative dormancy. The gradient-based nature of **MCP** makes it unique in discrete optimization settings. There is no other known approach to resolving mixed-integer linear decision scenarios that is based upon a gradient.

In my research and analysis of the **MCP** algorithm, I have made the following contributions:

1. I have established that **MCP.0** is non-convergent.
2. I have provided some key insights into the nature of the algorithm.
3. I have created and applied enhancements that define an original version of **MCP** that is theoretically both convergent and finite.
4. I have further enhanced **MCP** to make it more efficient with "multiple choice" problems.
5. I have demonstrated the efficiency of the extended algorithm in a number of test cases.

The basic flow of the modified algorithm, **MCP.1**, is as follows:

1. Employ the LP Simplex method to initialize **MCP**.
2. Eliminate non-integer optima with enhanced Profit Constraints, considering objective function granularity and known lower bounds to omit sub-optimal variables and redundant constraints.
3. Re-run the LP, noting greatest lower bound progress. Terminate when the

profit function value is at its greatest lower bound or if an LP solution is feasible.

4. If convergence failure occurs, disable variables irrelevant at the current profit function value and use Gomory as needed to eliminate non-integer optima.
5. Return to Step 2.

There are a number of techniques that should be explored to further improve the efficiency of **MCP**. Doing so may result in greatly improved performance, as well as suggest benefit from integrating **MCP** into other algorithms and/or general heuristic approaches. Among these approaches would be (A) employing industrial strength round-off error control, (B) increasing memory capacity and processing speed, (C) employing advanced methods of updating the simplex tableau to avoid the computational burden of restarting the LP from scratch each time and (d) employing more aggressive cutting plane techniques. I anticipate that these various approaches may yield optimal results in both a broader testing context and in practice.

When such an investment is made, significant benefit may eventually be derived from **MCP**, and from the proposed enhancements, in a variety of practical constrained mixed-integer linear decision scenarios in such diverse areas such as project scheduling, payload optimization, portfolio analysis, transportation optimization, plant design, blending, production scheduling and site location. **MCP.1** appears to be particularly efficient in resolving problems involving a number of multiple choice constraints. When problems are encountered that contain such constraints, employing **MCP.1** to resolve them may indeed yield significant benefit.

REFERENCES

1. Balas, E., *An Additive Algorithm for Solving Linear Programs With 0-1 Variables*, Operations Research, Vol. 13, No. 4, pp. 517-49 (1965)

2. Balas, E., Ceria, S., and Cornue 'Jols, G., Mixed 0-1 Programming by Lift-and-Project in a Branch-and-Cut Framework, , (1996)

3. Beasely, J.E., Advances in Linear and Integer Programming, Oxford University Press, New York, pp 187-221 (1996)

4. Bowman, V. and Starr, J, "Partial Orderings in Implicit Enumeration," Annals of Discrete Mathematics 1, Studies in Integer Programming, North Holland Publishing Co., New York, pp. 99-116 (1977)

5. Carnegie Mellon, http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/test/sac/sac94ste.tgz (1995)

6. Edgar, T.F. and Himmelblau, Optimization of Chemical Processes, McGraw Hill, New York, pp.397-423 (1988)

7. Freville, A. and Plateau, G. *Hard 0-1 Multi-Knapsack Test Problems For Size Reduction Methods.* Investigation Operativa, 1:251-270 (1990)

8. Garfinkel, R. and Nemhauser, G., Integer Programming, John Wiley & Sons, Inc. New York, pp 162-195 (1972)

9. Haldi, J. *25 Integer Programming Problems*, Working Paper No. 43, Graduate School of Business, Stanford University.

10. Healy,W.C. Jr., "Multiple Choice Programming", Management Science, vol 12, no. 1, pp: 122-138 (1964).

11. Martello, S., and Toth, P., Knapsack Problems: Algorithms and Computer Implementations, Wiley, Chichester, U.K., 108–109 (1990)

12. Petersen, C.C., *Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R&D Projects,* Management Science, Vol. 13, pp. 736-750. (1967)

13. Rice, www.caam.rice.edu/~bixby/miplib, (1996)

14. Roussos, G., M.S. Thesis, An Application of Multiple Choice Programming to Capital Budgeting Problems, M.S. Thesis, Colorado School of Mines, Golden, CO. (1995)

15. Senyu, S., and Toyada, Y., *An Approach to Linear Programming With 0-1 Variables,* Management Science, Vol. 15, pp.196-207. (1967)

16. Shi, W., *A Branch and Bound Method for the Multi-Constraint Zero One Knapsack Problem,* JORS., Vol. 30, pp.369-378 (1979)

17. Steiner, Peter O., *Choosing Among Alternative Public Investments In The Water Resource Field,* American Economic Review, Vol. 69, pp. 893-916.

18. Trauth, C., and Woolsey, R., "Integer Linear Programming: A study in Computational Efficiency", Management Science, Vol 15, No. 9, pp 481-493 (1969)

19. Van Moeseke, P. "An Algorithm for the Location of Central Facilities," Mathematical Programming for Activity Analysis, North Holland Publishing Co., New York, pp. 141-158 (1974)

20. WANG, X., A New Implementation of the Generalized Basis Reduction Algorithm for Convex Integer Programming. Ph.D. Dissertation, Yale University, New Haven, CT. (1997)

21. Weingartner, H.M., and Ness, D.N., *Methods For the Solution of the Multi-Dimensional 0/1 Knapsack Problem,* Operations Research, Vol.15, pp.83-103 (1967)

22. Woempner, M., Multiple Choice Programming, A Generalized Reduced Gradient Method for Solving Mixed Integer Problems with Zero-One and Continuous Variables, Ph.D. Dissertation, Colorado School of Mines, Golden, CO. (1994)

23. Woolsey, Robert E.D., *Dam Siting in the Delaware River Basin*, class notes, Integer Programming, Colorado School of Mines, Golden, CO (1993)

24. Woolsey, Robert E.D., Ph.D. Dissertation, An Application of Integer Programming To Optimal Water Resource Allocation For The Delaware River Basin, University of Texas at Austin, Austin, Texas, (1969)

25. Wu, N. and Coppins, R., Linear Programming and Extensions, McGraw-Hill Publishing Company, New York (1981)