SOME HEURISTICS AS PREPROCESSING FOR 0-1 INTEGER PROGRAMMING

A THESIS
SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCES
OF BİLKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Fatih YILMAZ

June, 1991

SOME HEURISTICS AS PREPROCESSING FOR 0-1 INTEGER PROGRAMMING

A THESIS

SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BİLKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Fatih Yılmaz

June, 1991

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Béla Vizvári(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Prof. Halim Doğrusöz

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Osman Oğuz

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Peter Kas

Approved for the Institute of Engineering and Sciences:

[illegible]

Prof. Mehmet Baray
Director of Institute of Engineering and Sciences

ABSTRACT

SOME HEURISTICS AS PREPROCESSING FOR 0-1 INTEGER PROGRAMMING

Fatih Yılmaz
M.S. in Operations Research
Supervisor: Assoc. Prof. Béla Vizvári
June, 1991

It is well-known that 0-1 integer programming is one of the hard problems to solve other than special cases of constraint set in mathematical programming. In this thesis, some preprocessing will be done to get useful informations, such as feasible solutions, bounds for the number of 1's in feasible solutions, about the problem. A new algorithm to solve general (nonlinear) 0-1 programming with linear objective function will be devoloped. Preprocessing informations, then, are appended to original problem to show improvements in enumerative algorithms, e.g. in Branch and Bound procedures.

Keywords: 0-1 Programming, Heurictic algorihm, Preprocessing.

ÖZET

0-1 TAMSAYILI PROBLEMLER ICIN BAZI SEZGISEL YONTEMLER

Fatih Yılmaz
Yöneylem Araştırması Yüksek Lisans
Tez Yöneticisi: Doç. Béla Vizvári
Haziran, 1991

0-1 tamsayı programlamaları, genelde çözumlenmesi zor problemlerdir. Eger sınır kumesi ozel bir hal gosteriyorsa, bu problem polinom zamanda çözen algorithmalar vardır. Bu tezde, problemin zorluğunuda düsünerek, bazi ön işlemler yapılacaktır. Sırasi ile, olurlu çözumler, herhangi bir olurlu cözümdeki 1 lere alttan ve de ustten sınır vermek gibi. Daha sonra, genel 0-1 programlamayı çözecek yeni bir algoritmanın tanıtımı yapilacaktır. Bu ön ışlemlerden cıkan sonuçları kullanarak, bırerleme algoritmalarında, örneğın dal ve sınır algorithmasında, yapılabilinecek iyileştirmelerden bahsedilecektir.

Anahtar kelimeler: 0-1 Programlama, Sezgisel algoritma, Ön ışlem.

To my parents,

ACKNOWLEDGEMENT

I would like to thank to Assoc. Prof. Béla Vizvári for his supervision, guidance, suggestions, and patience throughout the development of this thesis. I am grateful to Prof. Halim Doğrusöz, Assoc. Prof. Osman Oğuz and Assoc. Prof. Peter Kas for their valuable comments.

Also, I would like to thank to my love Yıldız, and members of my parents, Abdulkadir (father), Ayşe (mother), Deniz (sister) and Murat (brother) for their patience and love throughhout this thesis.

Special thanks are for my friends both in life and not in life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF TABLES

1. INTRODUCTION

A wide class of practical problems can be modelled using integer variables and linear constraints [6,7,12,15,17,19,21,30]. We can think of sitiuations where it is only meaningful to take integral quantities of certain goods such as cars, aeroplanes, cities or use of integral quantities of some resource such as men.

Most of the practical integer programming models restrict the integer variables to two values 0 or 1. For instance, Travelling Salesmen Problem, Matching problem, and so on. Such 0-1 variables are used to represent ' yes or no ' decisions.

In this thesis, we shall deal with these kind of pure 0-1 integer programming problems which can be formulated as:

$$\begin{aligned} max\, z &= cx \\ A'x &\leq b' \\ x &\in \{0,1\}^n \end{aligned} \qquad (BP)$$

with assumptions, $A' \in Z^{m \times n}$, $b' \in Z^m$, *and* $c \in N_+^n$.

It is well-known that unless the constraints have a special structure, (BP) is a hard problem to solve, because the problem is NP-complete in general. While linear programming problems involving thousands of constraints and variables can almost certainly be solved in a reasonable amount of time, a similar situations does not hold for integer programming problems.

To attack these problems, one can need preprocessing. It means to collect as much useful information about the problem as possible. These kind of information can be :

- **Generations of feasible solutions,**
- **Calculation of bounds for objective function value,**
- **Generation of new constraints, which either contains aggregated information of the original constraints or is algebrically independet from them,**

♯ of Const.	♯ of Var.	Without Preprocessing (CPU Time)	With Preprocessin (CPU Time)
15	30	8 min.	3 min.
15	30	12	5
20	50	2	.5
20	50	9	9
20	50	50	35
20	50	55	40
20	60	10	4
20	60	55	30

Table 1.1: Importance of Preprocessing

- **Estimation of number of 1's in feasible solutions,**
- **Obligatory fixing of variables, e.t.c.**

Any kind of enumeration method can be accelareted by these informations. The following computational results have been obtained with LINDO packages on a PC. In 'With Preprocessing' step, we append results of preprocessings to original problem such as objective function constraint, surrogate constraint, lower and upper bounds for the number of 1's in any feasible solution of (BP). It can be seen that, except one problem, improvements are significant.

In Chapter 2, a combined heuristic method to generate feasible solutions is devoloped. Lagrange problem, surrogate constraint problem and some heuristics will be combined to find feasible solutions efficiently in this algorithm. It will be shown that these solutions are reached in a very short time. And an algorithmic optimality test will be given. In Chapter 3, new bounding schemes for the number of 1's in feasible solutions of (BP) is discussed. The calculation of these bounds takes polynomial number of iterations. In Chapter 4, a feasibility test for a special system of 3 linear inequalities with 0-1 variables is constructed. Then, this test is applied to improve bounds given in Chapter 3. In Chapter 5, a new algorithm to solve general (non-linear) 0-1 programming with linear objective function is discussed. Finally, some conclusions are made.

2. GENERATION OF FEASIBLE SOLUTIONS

There are several ways for the use of Lagrange Multipliers (LM) in mathematical programming. Most of them is based on the paper, H.Everett [3]. The LM method is very simple, especially, in integer programming. A lot of papers are considering LM as a way to find a dual program for integer programming [22]. But LM is very useful algorithmic tool as well. It is used to decompose a large scale problem [6], or to reduce the number of variables [4]. The application of LM to generate feasible solutions and reduce the feasible region is discussed in [24].

In section 1, Lagrange problem, surrogate constraint problem and their relations are introduced [5,14]. In section 2, a combined heuristic to generate feasible solutions is devoloped, and an optimality test will be discussed. Some test problems which are randomly generated are devoted to see that how these feasible solutions are good.

2.1 Background review

Relaxation methods in integer programming are very important. Two of them, Lagrange relaxation and surrogate constraint problem, are discussed here.

Consider (BP), its Lagrange Relaxation Problem (LRP) is given as

$$L(\lambda') = \max\ _{x\in\{0,1\}^n} \{(c - \lambda' A)x + \lambda' b'\} \qquad (LRP)$$

where $\lambda' \geq 0$ is a fixed vector, and optimal solution can be found as:

$$x_j = \begin{cases} 1 & \text{if } c_j - \lambda' a'_j > 0 \\ 1 \text{ } or \text{ } 0 & \text{if } c_j - \lambda' a'_j = 0 \\ 0 & \text{if } c_j - \lambda' a'_j < 0 \end{cases}$$

where a'_j is the j^{th} column of A'.

Hence,

$$L(\lambda') = \sum_{j=1}^{n} |c_j - \lambda' a'_j|_+ + \lambda' b' \tag{2.1.1}$$

where $|\gamma|_+ = max\ \{0, \gamma\}$ for any arbitrary real number γ.

Lemma 2.1 [2] $\forall \lambda' \geq 0$, *(LRP) is an upper bound for (BP).*

H.Everett [3] gave in 1963 the main theorem of LM method, and then Nemhauser and Ulman [20] generalized his result a bit more.

Theorem 2.1 *If x^* solves problem (LRP) for a given $\lambda' \geq 0$ and x^* is feasible in (BP) and*

$$\lambda'(b - A'x^*) = 0 \tag{2.1.2}$$

then x^* solves problem (BP).

The surrogate constraint problem [8] is defined as:

$$\begin{aligned} &\max \quad cx \\ &\lambda' A' x \leq \lambda' b' \\ &x \in \{0,1\}^n \end{aligned} \qquad (SP(\lambda'))$$

where $\lambda' \geq 0$ is a fixed vector.

It can easily be seen that $SP(\lambda')$ is a relaxation of (BP).

Lemma 2.2 *$L(\lambda')$ is an upper bound of $SP(\lambda')$.*

Proof : Let x^* be an optimal solution of (LRP) for a given λ'. Then

$$L(\lambda') \geq cx + \lambda'(b' - A'x) \quad \forall x \in \{0,1\}^n$$

In addition,

$$\forall x \in SP(\lambda'), \quad \lambda'(b' - A'x) \geq 0. \quad \Box$$

Although $SP(\lambda')$ is a good relaxation to (BP), computationally, solving (LRP) is much more easier than $SP(\lambda')$, because $SP(\lambda')$ is a knapsack problem, therefore it is NP-hard.

2.2 A Combined Heuristic

The followings are some important properties of relaxation methods :

- solving relaxation problems are much more easier than original problem, these problems give an upper bound to original problem,
- if the optimal solution of these problems satisfying certain conditions is a feasible solution of the original problem, then it is optimal, too. E.g. in the case of (LRP), this extra condition is (2.1.2), as it can be seen from Theorem 2.1

Assume that a feasible solution, say, x^o is known. Then we can add (BP) the following objective function constraint :

$$cx \geq z_o + 1 \tag{2.2.1}$$

where z_o is the appropriate objective function value.

Then, it is enough to restrict ourselves to looking only for these feasible solutions, which have a better objective function value, i.e. the set of feasible solutions can be substituted by

$$P = \{x \;:\; Ax \leq b,\; x \in \{0,1\}^n\}$$

where

$$A = \begin{bmatrix} A' \\ -c \end{bmatrix}$$

and

$$b = \begin{bmatrix} b' \\ -z_o - 1 \end{bmatrix}.$$

instead of

$$P' = \{x \;:\; A'x \leq b',\; x \in \{0,1\}^n\}.$$

It is obvious that if the optimal solution is better than the given feasible solution, it is in P.

If we have initially no feasible solution, then without loss of generality we can assume that $cx \geq -1$, i.e. it is no restriction. Later on, if a feasible solution is found, the objective function constraint can be updated. The set P is used instead of P', i.e. the problem is considered in the form

$$\max \ cx$$
$$x \in P \qquad (MBP)$$

and (LRP) becomes,

$$L(\lambda) = \max_{x \in \{0,1\}^n} \{(c - \lambda A)x + \lambda b\} \qquad (MLRP)$$

where $\lambda = (\lambda', \lambda_{m+1}) \geq 0$.

In addition, $SP(\lambda')$ turns out to be

$$\begin{aligned} &\max \ cx \\ &\lambda A x \leq \lambda b \\ &x \in \{0,1\}^n. \end{aligned} \qquad (SP(\lambda))$$

In [24], a new optimality criterion was given. It defines the optimality region of a feasible solution, say, x^* generated by solving (MLRP), i.e,

$$arg \max_{x \in \{0,1\}^n} \{(c - \lambda A)x\} = x^*. \tag{2.2.2}$$

Then, the optimality region of x^* is given as:

$$H(x^*, \lambda) = \{x \in \{0,1\}^n \mid \lambda A x^* - \lambda A x \geq 0\} \tag{2.2.3}$$

Lemma 2.3 [24] *An optimal solution of the following system*

$$\begin{aligned} \max \quad & cx \\ & x \in P \cap H(x^*, \lambda) \end{aligned} \tag{2.2.4}$$

is x^*.

Lemma 2.3 is a good algorithmic tool for dividing feasible region in order to find optimal solution.

Let x^* generated by (MLRP) for a given $\lambda \in Z_+^{m+1}$, be a feasible solution to (MBP), and let define

$$P" = P \cap H(x^*, \lambda)^c \tag{2.2.5}$$

where

$$H(x^*, \lambda)^c = \{x \in \{0,1\}^n \mid \lambda A x^* - \lambda A x \ < \ 0\}.$$

Since all coefficients are integer, therefore if λ is an integer vector, then it is equivalent to

$$H(x^*, \lambda)^c = \{x \in \{0,1\}^n \mid \lambda A x^* - \lambda A x \ \leq -1\}. \tag{2.2.6}$$

Theorem 2.2 *Let x^* be the point generated by (MLRP) for a given $\lambda \geq 0$ and assume that it is a feasible solution of (MBP). If there is a feasible solution having better objective function value, then it is in P".*

Proof : By lemma 2.3, it is obvious. $\square$

Surrogate constraint problem is used to generate binary vectors. The following lemma contains optimality criterion of $SP(\lambda)$ for (MBP).

Lemma 2.4 *Let x^* be the optimal solution of $SP(\lambda)$. If x^* is in (MBP), then it is an optimal solution, too.*

$SP(\lambda)$ is a hard-problem to solve. Therefore a greedy algorithm is used to find a feasible solution of $SP(\lambda)$. One can write $SP(\lambda)$ more explicitely as

$$\begin{array}{ll} \max & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} \lambda a_j x_j \leq \sum_{i=1}^{m} \lambda_i b_i \\ & x_j \in \{0,1\} \quad \forall j \in J. \end{array} \tag{2.2.7}$$

To simplify the following steps, let us define

$$\beta = \sum_{i=1}^{m} \lambda_i b_i$$

and

$$\alpha_j = \lambda a_j.$$

Then, $SP(\lambda)$ becomes

$$\max \sum_{j=1}^{n} c_j x_j$$

$$\sum_{j=1}^{n} \alpha_j x_j \leq \beta$$

$$x_j \in \{0.1\}, \quad \forall j \in J.$$

In addition, without loss of generality we may assume that there is an index p, such that $\alpha_j \leq 0$ if $j \leq p$ and $\alpha_j > 0$ if $j > p$ and furthermore

$$c_{p+1}/\alpha_{p+1} \geq c_{p+2}/\alpha_{p+2} \cdots \geq c_n/\alpha_n. \tag{2.2.8}$$

In this ordering, α_j for some j can be equal to zero. But without losing this index ordering we can add a small quantity to denominators. Then, the following algorithm finds a feasible solution to $SP(\lambda)$.

Algorithm 1: A greedy algorithm for $SP(\lambda)$

```
 1. begin
 2.    for $j := 1$ to $p$ do
 3.       begin
 4.          $x_j := 1$;
 5.          $\beta := \beta - \alpha_j$;
 6.       end;
 7.    for $j := p + 1$ to $n$ do
 8.       if $\alpha_j \leq \beta$
 9.       then
10.       begin
11.          $x_j := 1$;
12.          $\beta := \beta - \alpha_j$;
13.       end;
14. end;
```

With this algorithm, we have x^* solution to $SP(\lambda)$ for a given $\lambda \geq 0$. Here the following question arises.

Are there Lagrange multipliers such that x^* is the optimal solution of the appropriate Lagrange problem ?

If the answer is **yes**, then all of the properties of points generated by Lagrange multipliers, e.g. the existence of the optimality region, can be preserved.

We will modify the result of [24], in such a way that λ vector which is used to find x^* is preserved here. In [24], $\lambda = 0$.

Let's extend (MBP) into

$$\begin{aligned} \max\ z = &\ cx \\ \\ \lambda:\ Ax \leq &\ b \\ \mu:\ Ex \leq &\ e \\ x \in &\ \{0,1\}^n \end{aligned} \tag{2.2.9}$$

where E is the appropriate identity matrix and λ and μ are respective Lagrange multipliers. Let $t \geq 0$ be a real number. Consider the following (MLRP) :

$$max_{x \in \{0,1\}^n} \{(c - t\lambda A - \mu)x\}. \tag{2.2.10}$$

Then previous question is transformed to
Are there t and μ such that

$$arg\ max_{\ x \in \{0,1\}^n} \{(c - t\lambda A - \mu)x\} = x^*$$

where x^* is the any feasible solution of $SP(\lambda)$?

Let k be defined as follows :

$$k = \max\{j\ :\ x_j^* = 1,\ and\ \alpha_j > 0\}. \tag{2.2.11}$$

If such k does not exists then there are two cases. The first one is that $\forall j\ \ \alpha_j \leq 0$. It means that all components of x^* is 1 . Due to cost coefficient, it is the optimal solution if feasible. The second case is that $\exists p < n$ with $\alpha_{p+1} > 0$. Then, let $k = p+1$ and we proceed as in the existence of k. So, choose

$$t = c_k / \lambda a_k \tag{2.2.12}$$

In addition,

$$\mu_j = \begin{cases} c_j - t\lambda a_j & \text{if } j < k \text{ and } x_j = 0 \\ 0 & \text{if not} \end{cases} \tag{2.2.13}$$

Lemma 2.5 *For t and μ chosen as above,*

$$arg\ \max_{\ x \in \{0,1\}^n} \{(c - t\lambda A - \mu)x\} = x^*.$$

Proof : If $t = c_k/\lambda a_k$, then

$$c_{p+1}/t\lambda a_{p+1} \geq \cdots \geq c_{k-1}/t\lambda a_{k-1} \geq 1 \geq c_{k+1}/t\lambda a_{k+1} \geq \cdots \geq c_n/t\lambda a_n. \quad (2.2.14)$$

Case 1: If $j = k$, then $c_k - t\lambda a_k - \mu_k = 0$,
Case 2: if $j > k$ and $\lambda a_j > 0, c_j - t\lambda a_j - \mu_j = c_j - t\lambda a_j < 0$,
Case 3: if $j < k$ and $x_j = 0$, we have $c_j - t\lambda a_j - \mu_j = c_j - t\lambda a_j - (c_j - t\lambda a_j) = 0$.
Case 4 : if $j < k$ and $x_j = 1$, then $c_j - \lambda a_j - \mu_j = c_j - \lambda a_j \geq 0$, due to (2.2.14).

□

Then, it follows that

$$cx^* - cx \geq (t\lambda A + \mu)x^* - (t\lambda A + \mu)x, \qquad \forall x \in \{0,1\}^n \quad (2.2.15)$$

In addition, the optimality region of x^* is

$$H(x^*, t\lambda, \mu) = \{x \in \{0,1\}^n \ : \ (t\lambda A + \mu)x^* - (t\lambda A + \mu)x \geq 0\} \quad (2.2.16)$$

Let us define

$$P" = P \cap H(x^*, t\lambda, \mu)^c \quad (2.2.17)$$

where

$$H(x^*, t\lambda, \mu)^c = \{x \in \{0,1\}^n \ : \ (t\lambda A + \mu)x_* - (t\lambda A + \mu)x \ < \ 0\} \quad (2.2.18)$$

In the applications, instead of (2.2.18) the following is used

$$H(x^*, t\lambda, \mu)^c = \{x \in \{0,1\}^n \ : \ (t\lambda A + \mu)x_* - (t\lambda A + \mu)x \leq -\epsilon\} \quad (2.2.19)$$

where $\epsilon > 0$ is a small number.

Lemma 2.6 *If there exists a feasible solution better than x^*, it is in P".* □

These methods, i.e. solving the Lagrange problem or the surrogate constraint problem, does not guarantee that feasible solution will be found at the first run. If it is not found, then λ vector will be modified. By these changes, the importance of constraints can be found. Importance means which constraints are much more difficult to satisfy. To collect this information, we can open an array such that each component of it corresponds the different constraint. Then if constraint i is infeasible for generated binary vector, then increase the vilolation number of this constraint by 1. The following modification of the Lagrange multiplier somehow reflect importance of the constraints.

Let x^* be the generated binary vector and assume that it is not feasible to (MBP). Then, the following sets are introduced

$$I(x^*) = \{i : a_i x^* > \ b_i\}$$

and

$$J(x^*) = \{j : a_j x^* \leq b_j\}.$$

Let us define

$$s_1 = \sum_{i \in I(x^*)} \lambda_i a_i x^*$$

$$t_1 = \sum_{i \in I(x^*)} \lambda_i b_i$$

and

$$s_2 = \sum_{j \in J(x^*)} \lambda_j a_j x^*$$

$$t_2 = \sum_{j \in J(x^*)} \lambda_j b_j.$$

Since $\lambda A x^* \leq \lambda b$,

$$s1 + s2 \leq t1 + t2$$

Let r be a real number such that

$$r > (t_2 - s_2)/(s_1 - t_1).$$

Choose new Lagrange multipliers [8] as

$$\lambda_k^{new} = \begin{cases} r\lambda_k & \text{if } k \in I(x^*) \\ \lambda_k & \text{if not} \end{cases} \tag{2.2.20}$$

The idea behind that choice is as follows: Assume that all of the components of the vector λ are positive and the vector x^* is generated by **Algorithm 1**. Then, x^* satisfies constraint sets of $SP(\lambda)$. But,

$$\sum_{i \in I(x^*)} \lambda_i a_i x^* > \sum_{i \in I(x^*)} \lambda_i b_i$$

and therefore

$$\sum_{j \in J(x^*)} \lambda_j a_j x^* < \sum_{j \in J(x^*)} \lambda_j b_j.$$

That kind of choice is a way to increase the weight of constraints which are not feasible. And the meaning of this kind of choice of r is described in the following lemma.

Lemma 2.7 *If r is chosen as in above and λ is modified as in (2.2.20), then x^* is not a feasible solution to $SP(\lambda^{new})$.*

Proof : If x^* is a feasible solution, then

$$r * s1 + s2 \leq r * t1 + t2$$

but this is a contradiction of

$$r \; > \; (t2 - s2)/(s1 - t1). \quad \square$$

In [8], for a given λ, $SP(\lambda)$ was solved optimally, and concentrated on the optimal solution of the (BP) because of Lemma 2.4. But, since $SP(\lambda)$ is a difficult problem, we are only concentrated on finding feasible solutions which are good and found in a short time.

Another heuristic to generate a feasible solution is as follows : Let x^o be any starting binary vector, e.g, choose x^o as a binary vector generated by Lagrange problem or surrogate constraint problem. Then, let

$$S(x^o) = \{y \in \{0,1\}^n \; : \; \sum_{j=1}^{n} |y_j - x_j| = 1\}$$

which is called the neighborhood set of x^o. The following function, g measures infeasibility or objective function value, resp., of a binary vector x if it is infeasible or feasible, resp., where

$$g(x) = \begin{cases} cx & \text{if } x \text{ is feasible} \\ \sum_{i=1}^{m} |b_i - \sum_{j=1}^{n} a_{ij}x_j| & \text{if not} \end{cases}$$

and the following set G contains all binary vectors in the neighborhood of x which are closer to be a feasible solution than previous ones;

$$G(x) = \{y \in S(x) \; : \; g(y) \; > \; g(x)\} \tag{2.2.21}$$

Then we get the following heuristic method to generate feasible solutions.

Algorithm 2

```
1. begin
2.     $k := 0$;
3.     Choose $x^o$;
4.     while $G(x^k) \neq \emptyset$ do
5.     begin
6.         Choose $x^{k+1} \in G(x^k)$;
7.         $k := k + 1$;
```

```
8.      end;
9. end;
```

There are several ways to execute the choice described in Row 6. One possible way is to choose the first neighbor which better than x^k.

These heuristics can be combined. The following algorithm is a general scheme for this combination, and computational experiences show that it is very effective to find feasible solutions if there are any.

Algorithm 3: A combined heuristic

```
1. begin
2.     $\lambda := \lambda^o$
3.     for $k := 1$ to $l$
4.     begin
5.         Apply Algorithm1; $x^k \in SP(\lambda^k)$    /* Binary Vector Generation */
6.         if $x^k \in P$
7.         then
8.             Put (2.2.19) into $P$    /* $P := P \cap H(x^k, t\lambda, \mu)^c$ */
9.         else
10.            Modify $\lambda$ vector as in (2.2.20);
11.    end;
12.    Apply Algorithm 2 with the starting point $x^l$;
13. end;
```

Now, we will give an optimality test. The following statement is an immediate consequence of the fact that the surrogate knapsack problem is a relaxation of the integer programming problem

$$if\ \ SP(\lambda) = \emptyset,\ \ then\ \ P = \emptyset.$$

For Algorithm 3, instead of arbitrary starting Lagrange multiplier, let us start with $\lambda^o = (\lambda', 0)$. Then, apply Algorithm 3 with the following rule :
RULE : Change the last component of λ , which corresponds to objective function constraint, to positive quantity **if and only if** Algorithm 3 finds a feasible solution.

With this choice of vector λ, the following theorem is valid:

Theorem 2.3 *With the above* **RULE**, *if $SP(\lambda) = \emptyset$ then, one of the following statement is true: 1) $P' = \emptyset$, i.e. original system is empty,*
2) let Y be the set of feasible solutions obtained by the Algorithm 3. Then

$$z = \max_{x \in Y} cx.$$

Proof : If $\lambda_{m+1} = 0$, i.e, no feasible solution is found, it follows that $SP(\lambda) = SP(\lambda')$. It means $SP(\lambda') = \emptyset$. Therefore, $P = \emptyset$, thus the problem is infeasible. On the other hand, if $\lambda_{m+1} > 0$, then we have found a feasible solution. But, $SP(\lambda) = \emptyset$. Due to Lemma 7 and Algorithm's property of adding constraint if we found a feasible solution, $P" = \emptyset$. But, $P"$ is used whenever there is a feasible solution. This completes the proof. □

In this theorem, there are two test which are infeasibility test and optimality test.
Test :

$$\textbf{Is } \sum_{j=1}^{n} \min\{0, \alpha_j\} \textbf{ greater than } \beta \text{ ?}$$

If the answer is yes and $\lambda_{m+1} = 0$, then original problem is infeasible, on the other hand, answer is yes and $\lambda_{m+1} > 0$, then an optimal solution is found.

2.3 Computational results

All problems are generated randomly. To find exact value of problems, LINDO is used. Tableau 1 is for the comparison of generated feasible solution value and exact value.

Computational experiences show that, for some problems, feasible solutions that we found, are the optimal solution. If the problem is infeasible , it can be detected. For the third problem, algorithm concludes that optimal solution is found. In general, difference between exact values and objective values of feasible solutions are small, and in computational point of view, it runs in a short time.

♯ of Const.	♯ of Var.	Density	♯ of Feasible sol.	Best found Obj.Fnc.Val	Infeas. Test	Optimal. Test	CPU time (in minute)	Exact Solu.
5	20	100	1	22	-	-	0.07 min.	22
5	20	80	1	22	-	-	0.04	22
5	20	60	1	53	-	+	0.003	53
10	20	80	-	-	+	-	0.005	-
10	20	60	1	17	-	-	0.07	21
10	20	60	1	16	-	-	0.008	17
10	20	40	1	13	-	-	0.05	15
15	30	80	-	-	+	-	0.09	-
15	30	60	1	24	-	-	0.10	27
15	30	40	1	38	-	-	0.12	47
20	50	100	1	41	-	-	0.30	43
20	50	80	1	26	-	-	0.25	48
20	60	80	1	37	-	-	0.34	38
20	60	40	1	28	-	-	0.70	42

Table 2.1: Result of Combined Heuristic

3. BOUNDING NUMBER OF 1's IN FEASIBLE SOLUTIONS

This part of the thesis refers to the set of the feasible solutions of 0-1 linear integer programming (MBP), i.e.

$$\begin{aligned} Ax &\leq b \\ x &\in \{0,1\}^n \end{aligned} \tag{3.0.1}$$

Generation of feasible solutions by heuristics are discussed in Chapter 2. This part of the thesis is to give bounds to the number of ones in feasible solutions [26,27,29].

In Section 1 of this chapter, two lower and upper bounds for the number of 1's in any feasible solution are discussed. Section 2 contains a numerical example. Then, in Section 3, experimental analysis is provided.

3.1 Bounding Schemes

The detailed form of (3.0.1) is

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i \qquad \forall i \in I$$

$$x_j \in \{0,1\} \qquad \forall j \in J.$$

where $I = \{1,\cdots,m\}$ and $J = \{1,\cdots,n\}$.

Define π_i to be a permutation over the set J for a given i∈I in which the coefficients of constraint i are sorted in nondecreasing order i.e

$$a_{i\pi_i(1)} \leq a_{i\pi_i(2)} \leq \cdots \leq a_{i\pi_i(n)} \tag{3.1.1}$$

There are three cases for any given constraint $i \in I$:

1. if $b_i \geq 0$, then the origin 0 is a feasible solution for that particular constraint,
2. if $\sum_{k \in N} a_{ik} > b_i$, where $N = \{k \ : \ a_{ik} < 0\}$, then there is no feasible solution for constraint i, therefore (BP) is infeasible.
3. if $\sum_{k \in N} a_{ik} \leq b_i$, where $N = \{k \ : \ a_{ik} < 0\}$, then there is at least one feasible solution for constraint i.

Further developments requires some definitions.

Definition 3.1 *$p_i(j)$ is the largest lower bound of the number of 1's that must be contained in any feasible solution of constraint i, if it exists with $x_j = 1$.*

Definition 3.2 *$q_i(j)$ is the smallest upper bound of the maximal number of 1's that can be contained in any feasible solution of constraint i, if it exists with $x_j = 1$.*

Assume that the indicies γ and θ defined by the following inequalities exist;
$\forall i < \gamma$,

$$a_{i\pi_i(j)} + \sum_{k=1, k \neq j}^{t} a_{i\pi_i(k)} \geq a_{i\pi_i(j)} + \sum_{k=1, k \neq j}^{\gamma-1} a_{i\pi_i(k)} > b_i \geq a_{i\pi_i(j)} + \sum_{k=1, k \neq j}^{\gamma} a_{i\pi_i(k)} \quad (3.1.2)$$

$\forall w > \theta$,

$$a_{i\pi_i(j)} + \sum_{k=1, \, k \neq j}^{w} a_{i\pi_i(k)} \geq a_{i\pi_i(j)} + \sum_{k=1, k \neq j}^{\theta+1} a_{i\pi_i(k)} > b_i \geq a_{i\pi_i(j)} + \sum_{k=1, k \neq j}^{\theta} a_{i\pi_i(k)} \quad (3.1.3)$$

i.e. γ is the smallest and θ is the greatest index t, such that

$$\sum_{k=1, \, k \neq j}^{t} a_{i\pi_i(k)} + a_{i\pi_i(j)} \leq b_i.$$

The shape of function $f(t) = \sum_{k=1}^{t} a_{i\pi_i(k)}$ is illustrated in Figure 1:

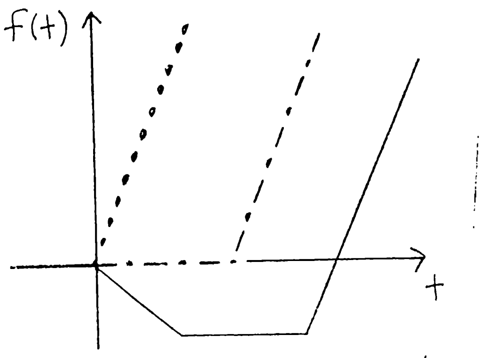


Figure 1: The shape of $\sum_{k=1}^{t} a_{i\pi_i(k)}$

In figure 1
.... represents a constraint i with positive coefficients,
-.-.-.- represents a constraint i with nonnegative coefficients,
—— represents a constraint i with negative, zero and positive coefficients.

The possible values of $p_i(j)$ and $q_i(j)$ in (3.1.2) and (3.1.3) are

$$p_i(\pi_i(j)) = \begin{cases} \gamma & \text{if a feasible solution with } x_{\pi_i(j)} = 1 \text{ exists} \\ n+1 & \text{if not} \end{cases}$$

and

$$q_i(\pi_i(j)) = \begin{cases} \theta & \text{if a feasible solution with } x_{\pi_i(j)} = 1 \text{ exists} \\ -1 & \text{if not} \end{cases}$$

Hence, for a given constraint i, we have $p_i(j) = n+1 \quad iff \quad q_i(j) = -1$. Thus, if $\forall j \in J, \quad \forall i \in I.\ p_i(j) \neq n+1$, then $p_i(j) \leq q_i(j)$.

The following statements are consequences of definitions; for any $i \in I$, if $j < k$ then $p_i(\pi_i(j)) \leq p_i(\pi_i(k))$. It follows that for the same indices, $q_i(\pi_i(j)) \geq q_i(\pi_i(k))$.

The following figures are showing all of the possible cases of constraint i with $x_{\pi_i(j)} = 1$. In fact, these figures are conceptual figures, but they give a good idea about $p_i(\pi_i(j))$ and $q_i(\pi_i(j))$ values.

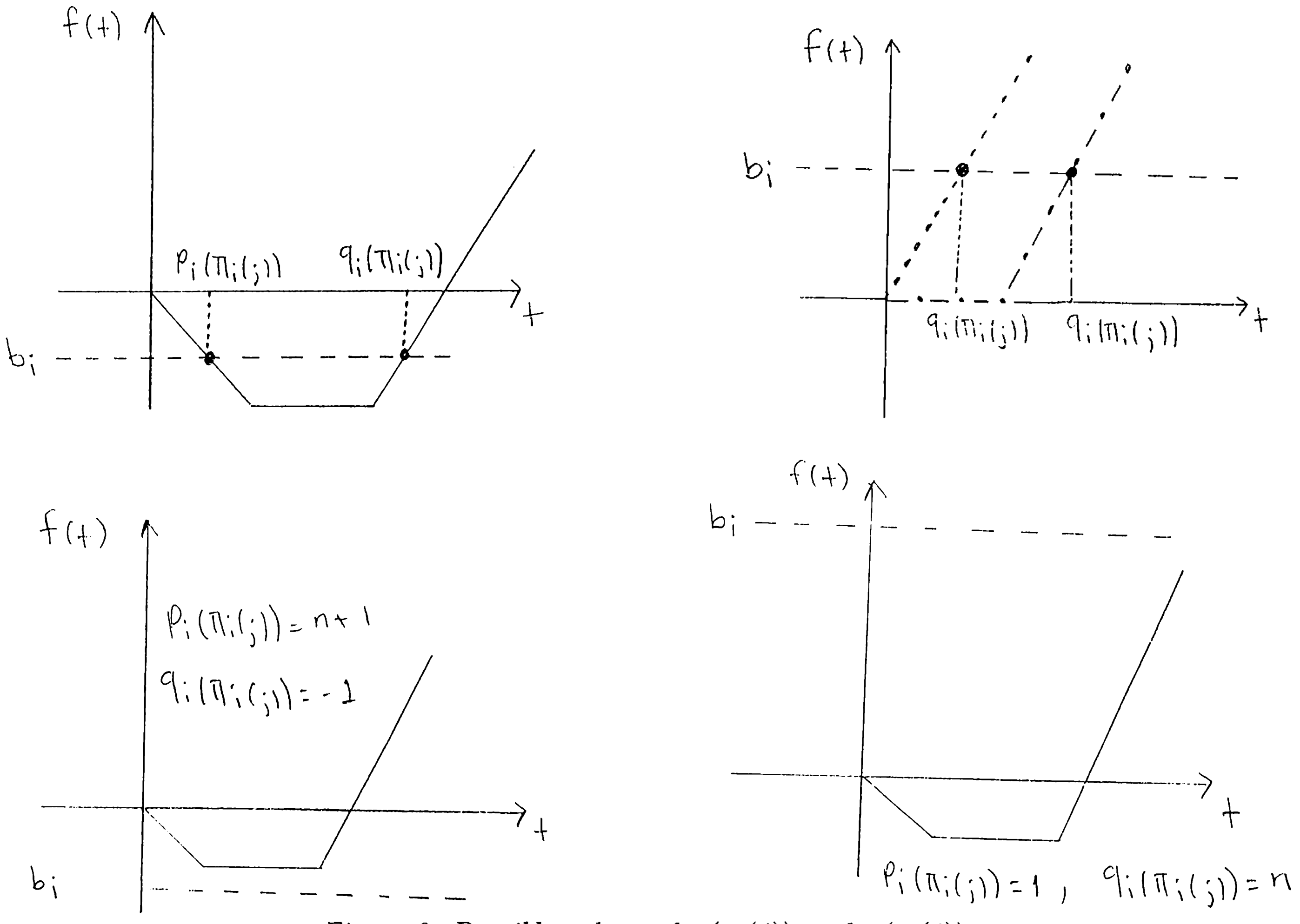


Figure 2: Possible values of $p_i(\pi_i(j))$ and $q_i(\pi_i(j))$.

Let $l(j)$ be the number of 1's which must be contained in any feasible solution of (3.0.1) if such exists with $x_j = 1$. Then

$$\forall j \in J, \qquad p(j) = \max{}_i \ p_i(j) \leq l(j). \tag{3.1.4}$$

Hence, for any feasible solution x with $x_j = 1$, we have

$$p(j) \leq \sum_{k=1}^{n} x_k. \tag{3.1.5}$$

Let $u(j)$ be the maximal number of 1's that can be contained in any feasible solution x with $x_j = 1$. It follows that

$$\forall j \in J, \qquad u(j) \leq q(j) = \min{}_i \ q_i(j). \tag{3.1.6}$$

and

$$\sum_{k=1}^{n} x_k \leq q(j). \tag{3.1.7}$$

Further on we use only numbers $p(j)$ and $q(j)$, instead of $l(j)$ and $u(j)$ respectively. Consider the index set

$$J(x) = \{j \ : \ x_j = 1\} \tag{3.1.8}$$

where x is a feasible solution. It follows that $|J(x)|$ is the number of 1's in x, i.e

$$\sum_{j=1}^{n} x_j = |J(x)|.$$

Lemma 3.1 *For any given feasible solution $x \neq 0$ to (3.0.1),*

$$\forall j \in J(x), \qquad p(j) \leq |J(x)|.$$

Proof : By definition of $p(j)$, it is obvious. □

The next theorem is the generalization of Lemma 3.1. It gives a lower bound for the number of 1's in any given feasible solution. We will call it as a trivial lower bound in the sense that only definition of $p(j)$ is used.

Theorem 3.1 *Let $x \neq 0$ be any feasible solution. Then,*

$$\sum_{j=1}^{n} x_j \geq \min{}_j \ p(j). \tag{3.1.9}$$

Proof : Let P be the set of feasible solutions of (MBP) By lemma 3.1, we have that

$$\forall x \in P \backslash \{0\}, \quad \forall j \in J(x), \qquad p(j) \leq |J(x)|.$$

Since $|J(x)| = \sum_{j=1}^{n} x_j$, it follows that

$$\min{}_j \ p(j) \leq \sum_{j=1}^{n} x_j, \qquad \forall x \in P \backslash \{0\}. \qquad \square$$

The next lemma is for upper bound which is also called trivial.

Lemma 3.2 *For any feasible solution x,*

$$q(j) \geq |J(x)| \quad \textit{if} \ \ j \in J(x).$$

Proof : The statement follows immediately from the definition of $q(j)$'s. □

Theorem 3.2 *For any feasible solution to (MBP),*

$$\sum_{j=1}^{n} x_j \leq \max{}_j \; q(j). \tag{3.1.10}$$

Proof : The proof is similar to the proof of the previous theorem. □

Now, we will give second lower and upper bounds for any feasible solution. These bounds are at least as good as obtained from Theorem 3.1 and Theorem 3.2.

Let's define L_l as the set of variables which can be contained with value 1 in at least one feasible solution if it exists with at most l 1's. In other words,

$$\forall l \in \{1, \cdots, n\}, \qquad L_l = \{j \ : \ l(j) \leq l\}$$

It follows from (3.1.6) that,

$$L_l \subseteq \{j \ : \ p(j) \leq l\} = \ T_l.$$

It is obvious that

$$T_1 \subseteq T_2 \subseteq \cdots \subseteq T_n.$$

Theorem 3.3 *For any feasible solution, $x \neq 0$, we have*

$$\sum_{j=1}^{n} x_j \geq \omega_1 \tag{3.1.11}$$

where

$$\omega_1 = \min\{l \ : \ |T_l| \geq l, \ l \geq 1\}.$$

Proof : Let us consider a feasible solution x with exactly l 1's in it. Then $\forall j \in J(x)$, $p(j) \leq l$. Hence, $J(x) \subseteq T_l$. Now, $l = \ |J(x)| \leq |T_l|$ and thus the statement follows. □

For a given $l \in \{1, \cdots, n\}$, define U_l such that it contains only variables which could be with value 1 in a feasible solution which solution contains at least l 1's, i.e,

$$U_l = \{j \ : \ u(j) \geq l\}$$

It follows from (3.1.6) that

$$U_l \subseteq \{j \ : \ q(j) \geq l\} = S_l.$$

It is obvious that

$$S_1 \supseteq S_2 \supseteq \cdots \supseteq S_n$$

Theorem 3.4 *For any feasible solution* x , *we have*

$$\sum_{j=1}^{n} x_j \leq \omega_2 \tag{3.1.12}$$

where

$$\omega_2 = \max\{l \ : \ |S_l| \geq l, \ l \geq 1\}.$$

Proof : The proof is similar to the proof of the last theorem. □

The following two algorithms are constructed to determine the values ω_1 and ω_2.

Algorithm ω_1

```
 1. begin
 2.     $count := -1$;
 3.     $l := 0$;
 4.     while $count < l$ do
 5.     begin
 6.         $l := l + 1$;
 7.         $count := 0$;
 8.         for $j := 1$ to $n$ do
 9.           if $p(j) \leq l$
10.           then
11.               $count := count + 1$;
12.     end;
13.     $\omega_1 = l$;
14. end;
```

Algorithm ω_2

```
1. begin
2.     $count := 0$;
3.     $l := n + 1$
4.     while $count < l$ do
5.     begin
```

```
6.          $l := l - 1$;
7.          count := 0;
8.          for $j := 1$ to $n$ do
9.            if $q(j) \geq l$
10.           then
11.             count := count + 1;
12.       end;
13.       $\omega_2 := l$
14.   end;
```

The following theorems and corrolaries are the consequences of the previous theorems and lemmas.

Theorem 3.5 *If $\omega_1 > \omega_2$, then (MBP) is infeasible.* □

Theorem 3.6 *If $p(j) = n + 1$, then for any feasible solution x, $x_j = 0$.* □

Corrolary 3.1 $\{j : q(j) = -1\} = \{j : p(j) = n + 1\}$. □

Corrolary 3.2 *If there exists an index j with $q(j) = n$, then e is a feasible solution,where e is all 1 vector.*

Proof : By definition of $q(j)$, $\forall i \in I$, $q_i(j) = n$, i.e. $\forall i \in I$, the vector consisting of n 1's is feasible. □

Corrolary 3.3 *If for an index j, $q(j) < p(j)$, then $x_j = 0$ in any feasible solution.* □

3.2 An Example

Consider the following system of linear inequalities:

$$\begin{aligned} -3x_1 + 4x_2 - 2x_3 + 5x_4 &\leq -2 \\ 2x_1 - x_2 + 4x_3 - 2x_4 &\leq 2 \qquad (P) \\ x_i \in \{0,1\},\ i &= 1,2,3,4 \end{aligned}$$

	x_1	x_2	x_3	x_4
$p_1(j)$	1	5	1	5
$p_2(j)$	1	1	2	1
$p(j)$	1	5	2	5
$q_1(j)$	2	-1	2	-1
$q_2(j)$	3	3	3	3
$q(j)$	2	-1	2	-1

Tableau 1: Results of computations

If we analyze Tableau 1, we can say that $x_2 = x_4 = 0$ due to Theorem 3.6. Moreover, $\omega_1 = min_j \ p(j) = 1$ and $\omega_2 = max_j \ q(j) = 2$. Therefore, by Theorem 3.3 and 3.4, we have

$$1 \leq \sum_{j=1}^{4} x_j \leq 2$$

. Hence,

$$1 \leq x_1 + x_3 \leq 2$$

If we rewrite P with $x_2 = x_4 = 0$, nothing is lost in the sense of feasible points. Then, new system is given by :

$$-3x_1 - 2x_3 \leq -2$$

$$2x_1 + 4x_3 \leq 2$$

$$x_1, x_3 \in \{0, 1\}$$

Then if the algorithm is applied again, then the following results are obtained.

	x_1	x_3
$p_1(j)$	1	1
$p_2(j)$	1	5
$p(j)$	1	5
$q_1(j)$	2	2
$q_2(j)$	1	-1
$q(j)$	1	-1

Tableau 2 : Result of second iteration of algorithm

Hence, $x_3 = 0$. The final result is that, the only feasible solution is (1,0,0,0).

3.3 Experimental Results

In this section, computational experiences are discussed. Tableau in Chapter 4 provides informations about ω_1, ω_2, exact lower and upper bounds. All of these problems are generated randomly. To find exact lower and upper bounds for the number of 1's that must be satisfied by any feasible solutions other than 0, LINDO package is used.

The calculation of ω_1 and ω_2 takes polynomial number of iterations. p and q vectors can be used in any enumeration algorithm for bounding number of 1's in any subset of feasible solutions within a very short time.

4. IMPROVEMENTS ON BOUNDS

This part of the thesis is devoted to improve bounds discussed in the previous chapter. In Section 1 of this chapter, a feasibility test for 3-linear constraints of 0-1 variables is given. Then, in Section 2, this test, for a special system, is going to be applied for sharpening the upper bound and improving the lower bound. One of these constraints is the objective function constraint, the second one is the surrogate constraint with special choice of Lagrange multipliers, and the third one is the constraint of the number of 1's that can be contained in some feasible solution. The last constraint has varying right hand side, say ω, such that $\omega \in [\omega_1, \omega_2]$. In Section 3, computational experiences will be discussed. Finally, some conclusions are obtained.

4.1 A Feasibility Test

Let us consider the set of binary vectors

$$P(\omega) \;=\; \{x \in \{0,1\}^n \;:\; \sum_{j=1}^{n} d_j x_j \geq \eta,\; \sum_{j=1}^{n} a_j x_j \leq \rho,\; \sum_{j=1}^{n} x_j = \omega\}.$$

where the two inequalities are consequences of the constraints of the (MBP) problem. If $P(\omega)$ is empty, then the (MBP) problem has no feasible solution with exactly ω 1's.

An easy test to check this is developed here. First of all, some preliminaries are required. Let us define

$$P_o(\omega) \;= \{x \in \{0,1\}^n \;:\; \sum_{j=1}^{n} a_j x_j \leq \rho,\; \sum_{j=1}^{n} x_j = \omega\}.$$

It is obvious that $P(\omega) \subseteq P_o(\omega)$. The following lemma will be called 'main lemma'.

Lemma 4.1 *If* $\exists y \in P_o(\omega)$ *with* $\sum_{j=1}^{n} d_j y_j \geq \eta$, *then* $P(\omega) \neq \emptyset$. □

Let us consider the following 0-1 Integer Programming problem:

$$\begin{aligned} \delta_\omega = \max \quad & \textstyle\sum_{j=1}^{n} d_j x_j \\ x \in \quad & P_o(\omega) \end{aligned} \tag{4.1.1}$$

The following lemma is equivalent to Lemma 4.1 in the sense that if one of them has a positive result, so has the other one. Positive result means $P(\omega) \neq \emptyset$.

Lemma 4.2 $\delta_\omega \geq \eta$ *iff* $P(\omega) \neq \emptyset$. □

Thus

$$P(\omega) = \emptyset \quad iff \quad \delta_\omega < \eta.$$

But (4.1.1) is a diffucult problem. Let us consider the LP relaxation of $P(\omega)$ and $P_o(\omega)$ as $P'(\omega)$ and $P'_o(\omega)$, i.e.

$$P'(\omega) = \{x : \sum_{j=1}^{n} d_j x_j \geq \eta, \ \sum_{j=1}^{n} a_j x_j \leq \rho, \ \sum_{j=1}^{n} x_j = \omega, \ 0 \leq x \leq e\}$$

and

$$P'_o(\omega) = \{x : \sum_{j=1}^{n} a_j x_j \leq \rho, \ \sum_{j=1}^{n} x_j = \omega, \ 0 \leq x \leq e\}$$

where e is the all 1 vector.

It is clear that $P(\omega) \subseteq P'(\omega)$ *and* $P_o \subseteq P'_o(\omega)$. Then, instead of (4.1.1), the following LP problem can be solved :

$$\begin{aligned} \delta'_\omega = max \quad & \textstyle\sum_{j=1}^{n} d_j x_j \\ x \in \quad & P'_o(\omega) \end{aligned} \tag{4.1.2}$$

(4.1.2) is an easy problem to solve. Its constraint set has special structure and upper bounding version of simplex method can be applied as a solution scheme very efficiently. Then, the following lemma arises :

Lemma 4.3 *If* $\delta'_\omega < \eta$, *then* $P(\omega) = \emptyset$.

Proof : If $\delta'_\omega < \eta$, $P'(\omega) = \emptyset$, but $P(\omega) \subseteq P'(\omega)$. □

Thus, a feasibility test is obtained for a system of 3 linear inequalities with 0-1 variables.

4.2 Application of Test

In Chapter 2, a combined heuristic which generate feasible solutions to (MBP) have been developed.

The following problem is identical to (MBP) :

$$\begin{aligned} max\ z = &\ cx \\ \lambda\ :\ Ax &\ \leq b \\ \mu\ :\ Ex &\ \leq e \\ x \in &\ \{0,1\}^n \end{aligned} \tag{4.2.1}$$

where E is the appropriate identity matrix, e is the all 1 vector, $\lambda\ \geq 0$, and $\mu \geq\ 0$ are appropriate Lagrange multipliers. Consider the following surrogate constraint problem :

$$SP(\lambda,\mu)\ :\ max\ Z =\ cx$$

$$(\lambda A + \mu)x \leq \lambda b + \mu e$$

$$x \in \{0,1\}^n.$$

Let x^o be a feasible solution of $SP(\lambda, 0)$ i.e.,

$$max\ Z =\ cx$$

$$\lambda Ax \leq \lambda b$$

$$x \in \{0,1\}^n$$

and assume that x^o is also a feasible solution of (MBP). Then, there exists t and μ (2.2.12-13) such that

$$argmax\ \{(c - t\lambda A - \mu)x\ :\ x \in \{0,1\}^n\}\ =\ x^o$$

in other words

$$max\ \{(c - t\lambda A - \mu)x\}\ =\ (c - t\lambda A - \mu)x^o.$$

Then, we have the following surrogate constraint :

$$(t\lambda A + \mu)x \leq t\lambda b + \mu e \tag{4.2.2}$$

It can be seen that $\forall x \in P$, x satisfies (4.2.2) . In addition, optimality region of x^o is defined as

$$(t\lambda A + \mu)x^o - (t\lambda A + \mu)x \geq 0.$$

It can be seen that if x^o is not an optimal solution of (MBP), then it must satisfy

$$(t\lambda A + \mu)x^o - (t\lambda A + \mu)x \leq -\epsilon \tag{4.2.3}$$

where $\epsilon > 0$ is the appropriate small number. The objective function constraint with this feasible solution, is defined as

$$cx \geq z_o + 1 \tag{4.2.4}$$

where $z_o = cx^o$.

If there is a feasible solution having better objective function value then z_o, it must satisfy (4.2.4). Furthermore, assume that we have an upper (lower) bound, say, ω, for the number of 1's in feasible solutions of P is known, i.e,

$$\sum_{j=1}^{n} x_j \leq (\geq)\omega. \tag{4.2.5}$$

Consider the following sets of binary vectors defined by linear inequalities :

$$P_1(\omega) = \{x \in \{0,1\}^n : \ cx \geq z_o + 1, \ \ (t\lambda A + \mu)x \leq t\lambda b + \mu e, \ \sum_{j=1}^{n} x_j = \omega\} \tag{4.2.6}$$

$$P_2(\omega) = \{x \in \{0,1\}^n : cx \geq z_o + 1, \ (t\lambda A + \mu)x^o - (t\lambda A + \mu)x \leq -\epsilon, \ \sum_{j=1}^{n} x_j = \omega\} \tag{4.2.7}$$

$$P_{1o}(\omega) = \{x \in \{0,1\}^n : \ (t\lambda A + \mu)x \leq t\lambda b + \mu e, \ \sum_{j=1}^{n} x_j = \omega\} \tag{4.2.8}$$

and

$$P_{2o}(\omega) = \{x \in \{0,1\}^n : \ (t\lambda A + \mu)x^o - (t\lambda A + \mu)x \leq -\epsilon, \ \sum_{j=1}^{n} x_j = \omega\} \tag{4.2.9}$$

Lemma 4.4 *If $P_1(\omega) = \emptyset$, or $P_2(\omega) = \emptyset$, then in any feasible solution with $cx > z_o$ we have $\sum_{j=1}^{n} x_j < w$ ($\sum_{j=1}^{n} x_j > w$).*

Proof : Since we assumed that $\exists\ x^*$ such that it is feasible to P and $cx^* \geq z_o + 1$. □

Let us modify problem (4.1.2) for these new constraints as

$$\begin{aligned} \delta'_\omega = \max \quad & \textstyle\sum_{j=1}^{n} c_j x_j \\ x \in \quad & P'_{1o}(\omega) \end{aligned} \tag{4.2.10}$$

$$\delta''_\omega = \max \quad \sum_{j=1}^{n} c_j x_j \qquad x \in \ P'_{2_o}(\omega) \tag{4.2.11}$$

where $P'_{1_o}(\omega)$ and $P'_{2_o}(\omega)$, resp., are the LP relaxations of $P_{1_o}(\omega)$ and $P_{1_o}(\omega)$, resp.

Then, the following algorithm gives sharpened upper bound of the number of 1's in any feasible solution.

Algorithm : Sharpening

```
1. begin
2.    $f := true$;
3.    $\omega := \omega_2$;
4.    while $f$ do
5.    if $\delta'_\omega$ $(\delta''_\omega)$ $<$ $z_o + 1$
6.    then
7.      $\omega := \omega - 1$
8.    else
9.      $f := false$;
10. end;
```

As a result of this algorithm and Lemma 4.4, the following corrolary is valid. It is a test for a given arbitrary ω that no need to search for larger ω values.

In above algorithm, if we choose ω as ω_1 at step 3 and $\omega = \omega + 1$ at step 7, lower bound ω_1 can be improved.

4.3 Computational results

All of the problems are generated randomly.. 'Before Improve.' column values are results of the algorithm that are found in Chapter 3. Last two columns is for the exact lower and upper bounds of some problems calculated by LINDO. Computational experiences shows that improvement are significant. Objective function and surrogate constraint are very effective over the upper bound, ω_2. Diference between exact upper bound and ω'' are, in general, very small. Only one problem has diference value as 4. Surrogate constraint, upto finding feasible solution to (MBP), collects aggregate information about the problem.

♯ of Const.	♯ of Var.	Before Improve.		After Improve.		Exact Lower Bound	Exact Upper Bound
		ω_1	ω_2	ω'	ω''		
5	20	1	11	7	9	1	9
5	20	2	11	4	7	4	6
5	20	2	15	11	15	2	14
10	20	2	14	4	8	2	7
10	20	2	16	6	13	2	12
15	30	1	23	11	21	1	17
15	30	1	25	10	18	1	16
20	50	6	25	12	22	6	21
20	50	5	32	11	21	6	20
20	50	6	31	8	17	6	17
20	60	1	36	10	22	2	22
20	60	2	37	11	24	3	21
20	60	4	49	11	36	4	35

Table 4.1: Lower and Upper Bounds

Special choice of Lagrange Multipliers, until generation of feasible solution, makes this information more compact.

Due to special structure of the constarint set of (4.2.10), computation of δ'_ω consumes very low CPU time, and above algorithm runs at most $\omega_2 - U$ times in which U is the exact upper bound.

5. A NEW ALGORITHM TO SOLVE GENERAL (non-linear) 0-1 PROGRAMMING

In this chapter, a new algorithm to solve general 0-1 programming problems with linear objective function is developed. Furthermore, this algorithm is adopted to solve linear 0-1 programming problems. The solution of the original problem, is equivalent with the solution of a sequence of set packing problems with special constraint sets. The solution of these set packing problems is equivalent with the ordering of the binary vectors according to their objective function value. An algorithm is developed to generate this order in a dynamic way. The main tool of the algorithm is a tree which represents the desired order of the generated binary vectors.

General 0-1 programming problem with linear objective function can be stated as:

$$\max \; cx$$

$$x \in S \subseteq \{0,1\}^n \tag{GBP}$$

where $c \in N_+^n$ and without loss of generality it is assumed in the whole chapter that

$$c_1 \geq c_2 \geq \cdots \geq c_n > 0. \tag{5.0.1}$$

It is well-known that the above problem (GBP) is a hard problem to solve if S has no special properties. If S is a set of linear inequalities with 0-1 variables, then it is (BP).

In [11] and [17], the following scheme is applied to solve the problem :

If $\beta_1 = e \in S$, then it is optimal where e is the all 1 vector. If $e \notin S$, then the optimal solution of (GBP), if it exists, must satisfy the following inequality

$$\sum_{j=1}^{n} x_j \leq n-1.$$

Let us consider the following problem :

$$\begin{aligned} Z_2 &= \max\ cx \\ & \textstyle\sum_{j=1}^{n} x_j \leq n-1 \\ & x \in \{0,1\}^n \end{aligned} \tag{5.0.2}$$

and assume that its optimal solution is β_2. Then, it is obvious that if $\beta_2 \in S$, then it is an optimal solution of (GBP). It can be seen that

$$\beta_{2j} = \begin{cases} 1 & if\ j \in \{1, ..., n-1\} \\ 0 & if\ j = n \end{cases}$$

Now assume that $\beta_2 \notin S$, it follows that optimal solution of original problem must satisfy

$$\sum_{j=1}^{n} x_j \leq n-1$$

$$\beta_2 x \leq \beta_2 e - 1$$

$$x \in \{0,1\}^n$$

If we continue this procedure, the following generalization is obtained.

Assume that $\beta_1 = e, \beta_2, ..., \beta_k$ have been generated so far and none of them was feasible to (GBP). Then, solve the following set packing problem :

$$Z_{k+1} = max\ cx$$

$$x \in P_k = \{x \in \{0,1\}^n \ : \ \beta_l x \leq \beta_l e - 1, \ l = 1, ..., k\} \tag{SPP}$$

and assume that the optimal solution of (SPP) is β_{k+1}.

Lemma 5.1 *Assume that $\beta_1, ..., \beta_k \notin S$. Then $S \subseteq P_k$.*

Proof: Let us assume that $y \in \{0,1\}^n$, $y \notin \{\beta_1, ..., \beta_k\}$, *but* $y \notin P_k$. Consider the following index r such that

$$r = min\{l \ : \ \beta_l y > \beta_l e - 1\}.$$

It is obvious that $r \geq 2$. Otherwise, $r = 1$, $y = \beta_1$. It follows that

$$\beta_r y = \beta_r e$$

i.e.,

$$y \geq \beta_r.$$

Hence, it follows from (5.0.1) that

$$cy > c\beta_r.$$

This is a contradiction to optimal property of β_r, because $y \in P_{r-1}$. □

Lemma 5.2 *(SPP) is a relaxation of (GBP).*

Proof: By previous lemma, $\forall x \in S,\ x \in P_k$. $\square$

Theorem 5.1 *Let k be the minimal index of iterations, such that $\beta_{k+1} \in S$. Then it is optimal to (GBP).*

Proof : It follows from previous lemmas. $\square$

5.1 Boros' Idea

It was Boros [1] who made the following observation in 1985. This solution scheme of sequence of set packing problems is equivalent to the ordering of the binary vectors in decreasing values of the objective function value. As it is stated in Lemma 1, the constraint set of the current set packing problem excludes only the points, generated so far. Therefore the optimal solution of next set packing problem is the best nongenerated binary vector.

Let L be the list that obeys above property. Then, the first feasible element of L is an optimal solution.
Example : let c be the following decreasing vector :

$$c = (6, 4, 3, 3)$$

Then, L contains
$L = \{(1,1,1,1), (1,1,1,0), (1,1,0,1), (1,0,1,1), (0,1,1,1). (1,1,0,0),$

$$(1,0,1,0), ..., (0,0,0,0)\}.$$

5.2 The Tree Representation of the Ordering of the Binary Vectors

In this section the following problem is discussed :

How can a sequence of the binary vectors be generated in which the values of the linear form cx are in a decreasing order ?

First of all, the subproblem is solved, where only the vectors having exactly in k components the value 1, i.e.

$$\sum_{j=1}^{n} x_j = k, \tag{5.2.1}$$

are ordered. Let u and v be two binary vectors satisfying (5.2.1). Let $\{i_1, i_2, ..., i_k\}$ and $\{j_1, j_2, ..., j_k\}$, resp., the set of indices of the 1's in the vector u and v, resp. It follows immediately from (5.0.1) that if

$$i_p \leq j_p, \quad p = 1, \cdots, k \tag{5.2.2}$$

then $cu \geq cv$.

Definition 5.1 *Let $\{i_1, ..., i_k\}$ be the set of indices of 1's in the binary vector u. Assume that for some index p the inequality*

$$i_p + 1 < i_{p+1}$$

holds, where $i_{k+1} = n + 1$. Then an immediate successor of u is the vector u', if

$$u'_j = \begin{cases} u_j & \textit{if } j \neq i_p, i_p + 1 \\ 0 & \textit{if } j = i_p \\ 1 & \textit{if } j = i_p + 1 \end{cases}$$

Any vector v satisfying (5.2.2) is a successor of u.

It is obvious from previous remarks, that u' is an immediate successor of u, then $cu \geq cu'$.

Theorem 5.2 *Let $\{i_1, ..., i_k\}$ and $\{j_1, ..., j_k\}$, resp., be the sets of indices of the 1's in the binary vectors u and v, resp. Assume that (5.2.2) holds. Then there is a sequence of binary vectors*

$$w_0 = u, \ w_1, ..., w_t = v, \tag{5.2.3}$$

such that w_l is an immediate successor of w_{l-1} $(l = 1, ..., t)$.

Proof : Assume that $u \neq v$, otherwise $t = 0$ and the statement holds. Let

$$p = max \ \{q \ : \ i_q \ < \ j_q\}.$$

Hence $u_{i_p+1} = 0$, otherwise

$$i_p + 1 = i_{p+1} \leq j_p \ < \ j_{p+1}$$

and this contradicts the maximal property of p. Let

$$w_{1j} = \begin{cases} w_{0j} & if\ j \neq i_p, i_p+1 \\ 0 & if\ j = i_p \\ 1 & if\ j = i_p + 1 \end{cases}$$

Then the set of the indices of 1's in w_1 is

$$\{i_{11}, ..., i_{1k}\} = \{i_1, ..., i_{p-1}, i_p + 1, ..., i_k\}.$$

It is obvious that

$$i_{1l} \leq j_l, \quad l = 1, ..., k.$$

Then either $w_1 = v$, or the process can be repeated for w_1. □

It is trivial that all of the binary vectors containing exactly k 1's is the successor of the vector x, where $x_1 = ... = x_k = 1$, $x_{k+1} = ... = x_n = 0$. Therefore all of these vectors can be enumerated with the following algorithm, where L is the list of binary vectors to be enumerated.

Algorithm 4

```
1. Begin
2.     $L := \{x = (1, ..., 1, 0, ...., 0)\}$;
3.     while $L \neq \emptyset$ do
4.     begin
5.         choose $u \in L$;
6.         $L := (L \cup \{v : v \text{ is an immediate successor of } u\}) \setminus \{u\}$;
7.     end;
8. end
```

Without any deeper organization this algorithm will work unnecessarily too much, because the sequence (5.2.3) is not unique, i.e. a binary vector can be generated several times as the immediate successor of different vectors. This phenomena is illustrated with the following example. Let $k = 2$, $n = 4$. Then Algorithm 4 starts from the point $x = (1, 1, 0, 0)$. In the first iteration it has only one immediate successor, which is $u_1 = (1, 0, 1, 0)$. The immediate successor of u_1 are $u_2 = (0, 1, 1, 0)$, and $u_3 = (1, 0, 0, 1)$. The only immediate successor of u_2 is $u_4 = (0, 1, 0, 1)$ which is at the same time immediate successor of u_3, too. To avoid this disadvantageous effect the following ordering of binary vectors is introduced.

Definition 5.2 *The vector u is 'greater' than v, denoted $u \rhd v$ if either*

- $cu > cv$ *or*
- $cu = cv$ *and u is lexicographically greater than v which is denoted by $u \succ v$.*

It is obvious that any two distinct vectors are comparable by this ordering, i.e. in any set of binary vectors there is a unique maximal element in this ordering. Thus the choice of the vector u in the 5-th row of Algorithm 4 can be executed in the following way

$$5' \quad u := max_{\,\unrhd} \{v \in L\}.$$

It follows from (5.0.1), that if u' is an immediate successor of u, then $u \rhd u'$. Hence the following statement is obtained.

Lemma 5.3 *If in Algorithm 4 the choice of the vector u is done according to 5', then no vector can be chosen twice.*

Proof : Assume that the vector u was chosen in iteration t. Then it was the maximal vector in ordering $\rhd$ which was contained in L and it was substituted by a set of smaller vectors. If it returned to L then at least one vector had to be substituted by a greater vector, which is impossible. □

Hence the following algorithm is obtained to enumerate all of the binary vectors. The vector e_j is the j-th unit vector.

Algorithm 5

```
1. Begin
2.    $L := \{\sum_{j=1}^{k} e_j \ : \ k = 1, ..., n\}$;
3.    while $L \neq \emptyset$ do
4.    begin
5.       $u := max_{\unrhd} \{v \in L\}$;
6.       $L := (L \cup \{v \ : \ v \text{ is an immediate successor of } u\}) \setminus \{u\}$;
7.    end;
```

```
8. end
```

Theorem 5.3 *All of the binary vectors are chosen in row 5 of Algorithm 5 exactly once.*

Proof : It follows from lemma 5.3 that it is enough to prove that all of the points are chosen at least once. Assume that the vector v containing exactly k 1's is not chosen. Let us consider a sequence (5.2.3) from the point $\sum_{j=1}^{k} e_j$ to v. Without loss of generality we may assume that in this sequence only v was not chosen. Then $w_t = v$ is an immediate successor of w_{t-1}, which was chosen in an iteration. When in row 5 u was w_{t-1}, w_t entered to L according to row 6. If v is in L and v has been never chosen then L has become never empty. Thus it follows from Lemma 5.3 that it should be infinite many binary vectors being greater in the ordering $\triangleright$ than v, which is a contradiction. □

For the sake of convenient handling of the list L it is organized as a rooted tree. The root contains always the greatest point. All of the nodes of the tree, except the root, can have two children, a left and a right one. The root has only a left child. All of the binary vectors of the left (right) subtree of a node are less (greater) than that of v according to the relation $\triangleright$. Thus it is very easy to find the appropriate position of a newly generated binary vector in the tree. This is done by Algorithm 6.

Algorithm 6

```
1.  begin
2.      (* Let L be the current tree *)
3.      (* S := Set of all immediate successor of x *)
4.      x := Binary vector in the root;
5.      while $S \neq \emptyset$ do
6.      begin
7.          $y \in S$;
8.          $z := cy$;
9.          $p := root^{\wedge}.left$;
10.         $f := true$;
11.         while $p \neq nil$ and $f$ do
```

```
12.      begin
13.        $Temp := p$;
14.        if $z > p^\wedge.obfv$
15.        then $p := p^\wedge.rightnode$
16.        else if $z < p^\wedge.obfv$
17.            then $p := p^\wedge.leftnode$
18.            else if $y \succ p^\wedge.BVector$;
19.                then $p := p^\wedge.rightnode$;
20.                else if $p^\wedge.BVector \succ y$
21.                    then $p^\wedge.leftnode$
22.                    else
23.                    begin
24.                      $f := false$;
25.                      $S := S - \{y\}$;
26.                    end;
27.     end;
28.     if $f$
29.     then
30.     begin;
31.        $S := S \backslash \{y\}$;
32.        if $z > Temp^\wedge.obfv$
33.        then $Temp^\wedge.rightnode := y$
34.        else if $z < Temp^\wedge.obfv$
35.            then $Temp^\wedge.leftnode := y$
36.            else if $Temp^\wedge.BVector \succ y$
37.                then $Temp^\wedge.leftnode := y$
```

```
38.                 else $Temp^\wedge.rightnode := y$;
39.     end;
40. end;
```

In the row 6 of Algorithm 5, the point contained in the root is omitted from the tree. Therefore a method is needed to find the point, which must go into the root, i.e. the maximal binary vector in the tree. But, notice that this is always the most right point of the (left) subtree of the root.

As a implementation of the above algorithm, let L_k be the current tree at step k, and assume that x_k, binary vector in the root, is not feasible to (GBP), otherwise it is optimal, and S is the set of immediate successor of x_k. Then

$$L_{k+1} := L_k \cup S \setminus \{x_k\} \tag{5.2.4}$$

Example : Assume that $n = 6$, $c = (7, 5, 4, 3, 2, 1)$ and the current tree L_k consists of the following binary vectors

$$x_k = (1, 0, 1, 0, 1, 0)$$

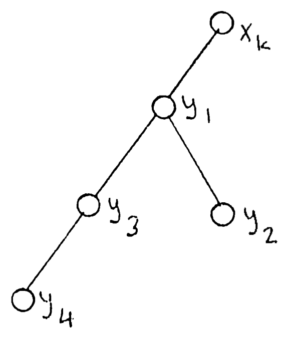
$$y_1 = (1, 0, 1, 0, 0, 0)$$

$$y_2 = (1, 1, 0, 0, 0, 0)$$

$$y_3 = (0, 1, 0, 1, 1, 0)$$

$$y_4 = (1, 0, 0, 0, 0, 0)$$

and the tree representation is



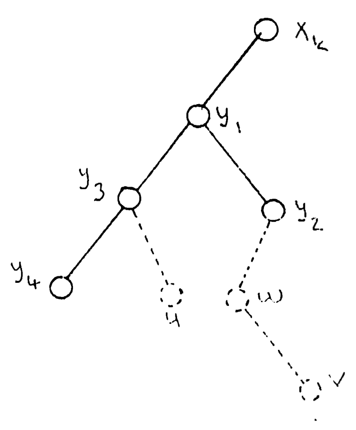
then immediate successor of x_k are

$$u = (0, 1, 1, 0, 1, 0)$$

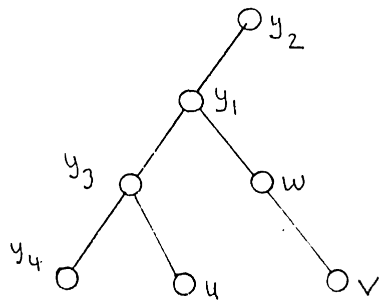
$$v = (1, 0, 1, 0, 0, 1)$$

$$w = (1,0,0,1,1,0)$$

then tree becomes



Then point y_2 goes into the root and the shape of tree is



In Chapter 3 and 4, the lower and upper bounds were given for the number of 1's in any feasible solution of 0-1 integer programming. If (GBP) is assumed to be 0-1 linear integer programming, these bounds can be used to make improvements in the algorithm.

5.3 An Example

Consider the following problem

$$max\ \ 2x_1 + x_2 + 3x_3 + 4x_4$$

$$3x_1 + 2x_2 + x_3 + 4x_4 \leq 3$$

$$2x_1 + 4x_2 + 2x_3 + 3x_4 \leq 6 \qquad (P)$$
$$x_1,\ x_2,\ x_3,\ x_4 \in \{0,1\}.$$

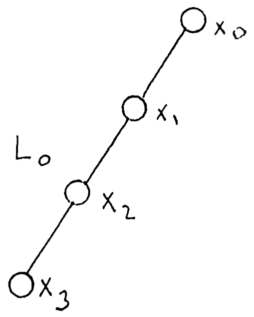
then

$$c_4 \geq c_3 \geq c_1 \geq c_2$$

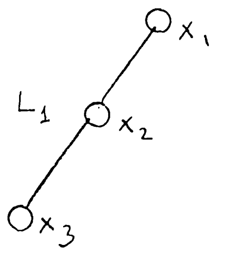
Therefore, index set used to generate immediate successors of binary vector in the root is (4,3,1,2). So, for the initial tree (L_o), the following binary vectors is used.

$$\{(1,1,1,1),\ \ (1,0,1,1),\ \ (0,0,1,1),\ \ (0,0,0,1)\} = \{x_0, x_1, x_2, x_3\}$$

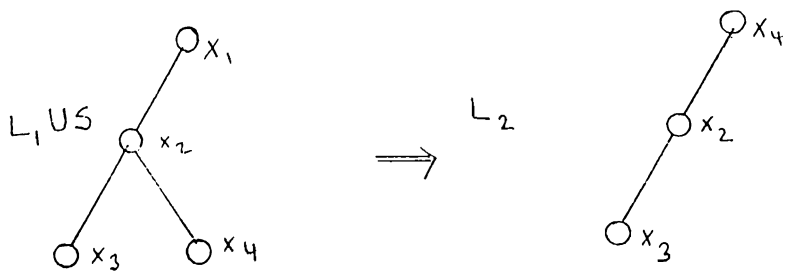
and tree becomes



$(1,1,1,1) \notin P$ then tree turns out to be

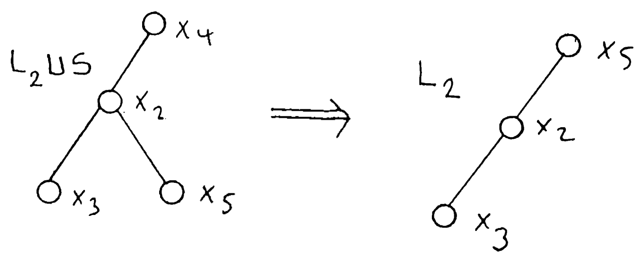


Binary vector in the root is $(1,0,1,1) \notin P$ and its ordering with respect to objective function value is $(1,1,1,0)$. Immediate successor of it in this ordering is $(1,1,0,1)$ and corresponding binary vector is $x_4 = (0,1,1,1)$. If we append it to the current tree, then tree becomes

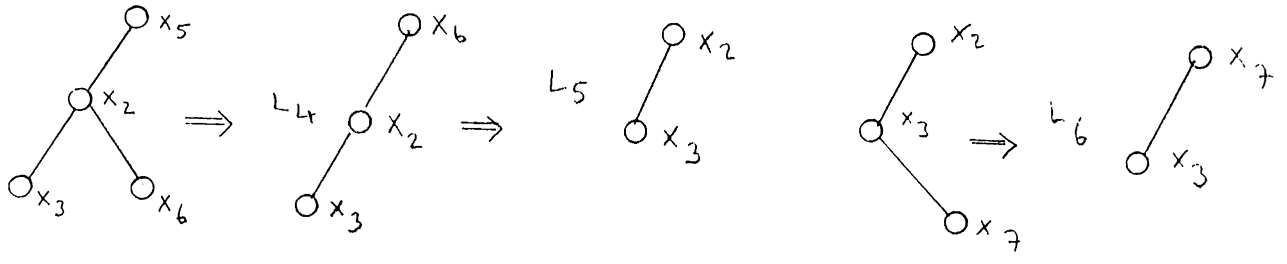


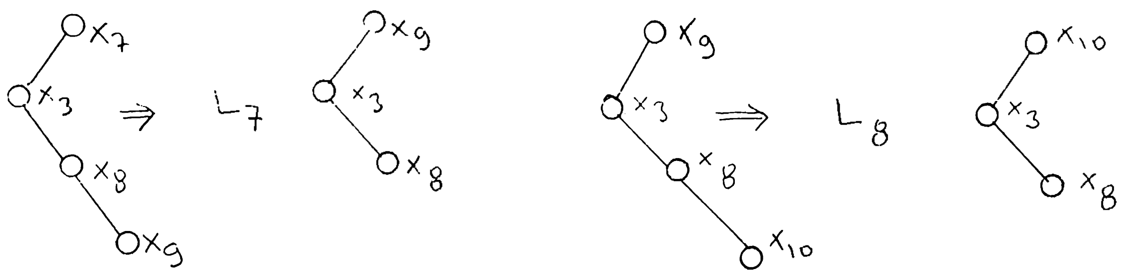
The binary vector in the root is $(0,1,1,1) \notin P$ and its ordering is $(1,1,0,1)$. Then the immediate successor of $(1,1,0,1)$ is $(1,0,1,1)$ and corresponding binary vector is

$x_5 = (1,1,0,1)$. If it is put in tree, tree becomes



The following iterations is similar and these are shown without explanations





where $x_6 = (1,1,1,0)$, $x_7 = (1,0,0,1)$, $x_9 = (0,1,0,1)$, $x_{10} = (0,1,1,0)$.

Now, we shall use informations, the bounds for the number of 1's , found in Chapter 3 and 4. The followings are the p and q vectors (see, Chapter 3)

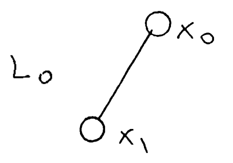
$$p = (1,1,1,5), \quad q = (2,2,2,-1)$$

It follows that $x_4 = 0$ by Theorem 3.6. Then $S\ (x_4 = 0)$ is

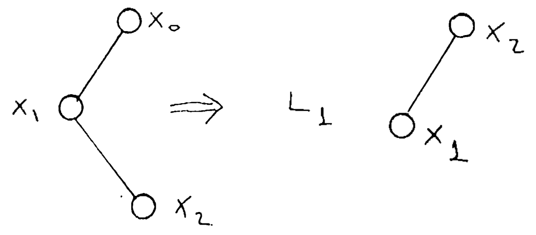
$$3x_1 + 2x_2 + x_3 \leq 3$$

$$2x_1 + 4x_2 + 2x_3 \leq 6$$

so if we calculate p and q vectors, then $p = (1,1,1)$, $q = (2,2,2)$. Then, no need to search for the subproblems having total number of 1's greater than two 1's. In addition, ordering (4,3 1 2) is reduced to (3.1.2). So, initial tree is



Binary vector in the root is $(1,0,1,0) \notin P$, and its ordering is $(1,1,0)$ Immediate successor of $(1,1,0)$ is $(1,0,1)$ and corresponding binary vector is $(0,1,1,0)$. Then tree becomes



Since binary vector in the root, $(0,1,1,0)$ is a feasible solution to P, then stop, it is optimal solution.

Computational experiences show that for some problems, tree representation of the ordering of the binary vectors for decreasing values of objective function value, solves the problem in a reasonable amount of time. Assuming that constraint set of (GBP) is difficult to handle by known algorithms, it generates upper bounds to (GBP) and generates thousands of binary vectors in a vey short time.

6. CONCLUSIONS

We have studied 0-1 linear integer programming problems. These optimization problems are usually NP-complete, except the specific cases that can be solved polynomially. Preprocessing is used to increase the effectiveness of the methods designed to solve these problems. The aim of the preprocessing stage is to collect as many useful information about the problem as possible. Preprocessing reduces computational times of the optimization methods. In this study, various preprocessing schemes were developed.

First of all a combined heuristic method was constructed to find good feasible solutions in a short time. Then, new lower and upper bounds were given for the number of 1's in feasible solutions. Morever, these bounds were improved by a special feasibility test for 3-linear inequalities with 0-1 variables. These inequalities were consequences of (MBP). Finaly, a new algorithm to solve general 0-1 programming with linear objective function were devoloped. And the results of the previous chapters were used to make refinements in this algorithm.

In any solution scheme developed to solve (MBP), such as a branch and bound, the use of the preprocessing improves the solution time (see Table in Chapter 1).

As an essential future of the study is to develope a computer code which combines and modifies the results of the thesis to solve large 0-1 integer programming problems within one system.

REFERENCES

[1] Boros, E., Private Comminication, 1985.

[2] Balas, E., Zemel, E., Solving Large Zero-One Knapsack Problems. Oprns. Res. 28 (1980) 1130-1154.

[3] Everett, H., Generalized Lagrange Multiplier Method for Solving of Optimum Allocation of Resources, Oprns. Res. (1963) 399-417.

[4] Fayard, M.L., Plateau, G., Resolution of the 0-1 Knapsack Problem : Comparision of Methods, Math. Programming, 8 (1975) 272-307.

[5] Fisher, M.L., The Lagrangian Relaxation Method of Solving Integer Programming Problem, Man. Sci. 27, (1981) 1-18.

[6] Fisher, M.L., Optimal Solution of Scheduling Problems Using Lagrange Multipliers : Part 1, Oprns. Res., 21 (1973) 1114-1127.

[7] Garfinkel, R.S., Nemhauser, G.L., Optimal Political Districting by Implicit Enumeration Techniques, Man. Sci. 16, (1970) B495-B508.

[8] Glover, F., A Multiphase Dual Algorithm for Zero-One Integer Programming, Opsn. Res., 13, (1965) 879-919.

[9] Glover, F., Surrogate Constraint Duality in Integer Programming, Opns. Res., (1975) 434-451.

[10] Glover, f., Woolsey, R.E., Converting the 0-1 Polynomial Programming Problem to a 0-1 Linear Program, Opns. Res. 22(1), (1974) 141-161.

[11] Granot, D., Granot, F., Kalberg, J., Converting Relaxation for Positive 0-1 Polynomial Programs, Management Science, 25, (1979) 264-273.

[12] Hirsh, W.M., Dantzing, G.B., The Fixed Charge Problem, Nav. Res. Log. Quart. 15, (1968) 413-424.

[13] Ingargiola, J.P., Korsch J.F., A Reduction Algorithm for Zero-One Single Knapsack Problems, Mng. Sci. 20 (1973) 460-463.

[14] Karwan, M.H., Rordan, R.L., Some Relationships Between Lagrangian and Surrogate (constraint) Duality in Integer Programming, Math. Prog. 17, (1979) 320-334.

[15] Koopmans, T.C., Beckmann, M.J., Assignment Problems and the Location of Economic Activities, Econometrica 25, (1957) 53-76.

[16] Lawler, E.L., Bell, M.D., A Method for Solving Discrete Optimization Problems, Opns. Res. 14(6), (1966) 1098-1112.

[17] Maga, F., Vizvári, B., The Relaxation of a Special Polynomial Zero-One Programming Problem to Set Covering Problem, Alkalmazott Matematikai Lapok, 12 (1986) 41-49.

[18] Manne, A.S., On the Job-Shop Scheduling Problem, Opns. Res. 8, (1960) 219-223.

[19] Murty, K.G., Solving the Fixed Charge Problem by Ranking the Extreme Points, Opns. Res. 16, (1968) 268-279.

[20] Nemhauser, G.L., Ullman, Z., A Note on the Generalized Multiplier Solution to an Integer Programming Problem, Operations Research 16, (1968) 450-452.

[21] Pritsker, A.A., Watters, L.J., Wolfe, P.M., Multiproject Scheduling with Limited Resources: A 0-1 Programming Approach, Man. Sci. 16, (1969) 93-108.

[22] Shapiro, J.F., A Survey of Lagrangean Techniques for Discrete Optimization, Annals of Discrete Mathematics, 5 (1979) 113-138.

[23] Taha, H.A., Further Improvements in the Polynomial Zero-One Algorithm, Man. Sci. 19(2), (1972) B226-227.

[24] Vizvári, B., Lagrange Multipliers in Integer Programming, Problems of Control and Information Theory, Vol.7 (5), 393-406 (1978).

[25] Vizvári, B., The Heuristic Method of Discrete Programming I: MTA SZTAKI, Tanulmanyok, 152 (1983) 109-138.

[26] Vizvári, B., Yılmaz, F., New Bounding Schemes for 0-1 Knapsack-Like Constraints, Research Reporth: 9021, Bilkent University, Turkey(1990).

[27] Vizvári, B., Yılmaz, F., Sharpening of the Number of 1's in Feasible Solutions of 0-1 Integer Programming, Research Report: 9104, Bilkent University, Turkey(1991).

[28] Vizvár, B., Yılmaz, F., A New Algorithm to Solve General 0-1 Programming Problem with Linear Objective Function, Research Report: 9105, Bilkent University, Turkey(1991).

[29] Yılmaz, F., Combined Heuristic Method for 0-1 Programming, Research Report: IEOR-9101, Bilkent University, Turkey(1991)

[30] Weingartner, H.M., Capital Budgeting of Interrelated Projects: Survey and Synthesis, Man. Sci. 12, (1966) 485-516.