International Journal of Academic Research in Business and Social Sciences
January 2012, Vol. 2, No. 1
ISSN: 2222-6990

A Low-Order Knowledge-Based Algorithm (LOKBA) to Solve Binary Integer Programming Problems

S. H. Pakzad-Moghadam

MSc student, University of Tehran, Industrial Engineering Department

E.mail: hojatpakzad@gmail.com

M. S. Shahmohammadi

BSc student, University of Tehran, Industrial Engineering Dept,
E.mail: sadegh_shahmohammadi@yahoo.com

R. Ghodsi

Assistant Professor, University of Tehran, Industrial Engineering Dept,
E.mail: ghodsi@ut.ac.ir

Abstract

In this paper a novel and very fast low order knowledge-based algorithm (called LOKBA) is presented to solve binary integer programming (BIP) problems. The factors in the objective function and the constraints of any BIP problem contains specific knowledge and information that can be used to better search the solution space of the problem. In this work many new definitions are introduced to elaborate on this information and extract necessary knowledge for creating and improving solutions. Found solutions are improved further and further until there is no new knowledge that can be extracted and there are no more possible improvements. The proposed algorithm has many major credentials. It produces promising results within amazingly short run times even for very large problems such as problems with one million variables. Also, it is extendable to solve integer programming and binary quadratic programming problems. Furthermore, it can be combined with heuristic or meta heuristic algorithms.

Keywords: *Binary Integer Programming, Combinatorial Optimization, Rule-based Algorithm, Mathematical Programming, Mixed Integer Programming*

1. Introduction

Binary integer programming (BIP) problems are a subset of Mixed Integer Programming (MIP) problems. Since there exist many practical applications such as distribution networks, scheduling in airline and other transportation industries, capital budgeting, telecommunications, sequencing and selection decisions, many researches focus on BIP

International Journal of Academic Research in Business and Social Sciences
January 2012, Vol. 2, No. 1
ISSN: 2222-6990

specifically as a major subset of MIP. The general BIP problem can be formulated as below:

$$\text{Max } \mathbf{cx}$$
$$\text{Subjected to } \quad \mathbf{Ax} \leq \mathbf{b}$$
$$\mathbf{x} \in \{0, 1\}^n$$

where **A** is an *m* by *n* matrix, **c** is an n-dimensional row vector, **b** an m-dimensional column vector, and **x** is an n-dimensional column vector of variables or unknowns. All integer programming problems are known to be NP-hard and difficult to solve and have long been an important research area (Wang and Xing, 2009) (Wolsey, 1998). Kellerer *et al* (2004) have provided a full-scale presentation of all methods available for the solution of the Knapsack problem as a BIP problem in their book (Kellerer et al., 2004). Because large size NP-hard problems are not amenable to solution by exact methods, generally heuristic algorithms are used to solve integer programming problems have been solved by different exact and heuristic methods. Exact methods include branch and bound, dynamic programming, Lagrangian relaxation based methods and integer programming based methods such as branch and cut, branch and price, and branch and cut and price (Nemhauser and Wolsey, 1988). Heuristics include simulated annealing (Kirkpatrick et al., 1983), tabu search (Glover and Laguna, 1997), population-based models such as evolutionary algorithms (Baeck et al., 1997), scatter search (Glover et al., 2000), memetic algorithm (Moscato and Cotta, 2003), various estimation of distribution algorithms (Larranaga and Lozano, 2001), etc. There are also many attempts that combine exact and heuristic methods (Domitrescu and Stuetzle, 2003). Some recent improvements in heuristics are Local Branching (LB) (Fischetti and Lodi, 2003), Variable Neighborhood Search and Local Branching (VNSB) (Hansen et al., 2006), Variable Neighborhood Decomposition search for 0-1 Mixed Integer Programs (VNDS) (Lazic et al., 2009), Relaxation Induced Neighborhood Search (RINS) (Danna et al., 2005) and Distance Induced Neighborhood Search (DINS) (Ghosh, 2007). Namazifar and Miller (2008) presented a framework called macro partitioning (PMaP) for solving MIP problems in parallel (Namazifar and Miller, 2008). Chu and Beasley (1998) have developed a genetic algorithm for the multidimensional knapsack problem (Chu and Beasley, 1998). Captivo et al. (2003) have used a labeling algorithm to solve bi-criteria zero-one Knapsack problems (Captivo et al., 2003). General BIP problems include both negative and non-negative coefficients. A lot of research work exists on Knapsack problems or problems with non-negative coefficients. Fortin and Tseveendorj (2009) presented a branch and bound procedure based on empirical distribution of each variable under the linear relaxation model (Fortin and Tseveendorj, 2009). Vasquez and Hao (2001) developed a hybrid approach for the zero-one multidimensional Knapsack problem and stated that most efficient algorithms rely on linear relaxation bounds coupled with extra constraints (Vasquez and Hao, 2001). Haartman et al. (2008) find approximate solutions to BIP problems using continualization techniques (Haartman et al., 2008). Their new algorithm constructs a sequence of approximations to a solution using a meta-control approach that calculates the least squares of the errors.

In the work at hand, both negative and non-negative coefficients are considered and none of the constraints is relaxed. The proposed algorithm considers both the objective and the constraints simultaneously and finds the solution very fast after some iteration. In each of the iterations the variables and constraints of the problem are inspected to draw knowledge and some of them are distinguished as “firm” or “strong” variables and “obsolete” constraints. The novel part of the algorithm is the drawing knowledge from the variables and constraint factors that is called “standardization” in this work. Based on that knowledge it then classifies the constraints and variables as mentioned in an innovative manner. Next the constraints and variables are weighted using a specific weighting scheme. Using these weights, the values for a few of the variables are specified and a smaller problem is then formed. Then the variables and constraints are re-inspected and new weights are calculated. In each of these steps, a few more of the variables are specified until all variables are known and the algorithm terminates. The algorithm is explained in detail in the next section. The validation of the algorithm is done through solving many different sample problems. The results were very promising and the optimum or good solutions were found in very short run time even for large size problems. For example, in a problem with 10,000 variables the optimum solution was found in less than 4 seconds for a normal PC.

The novel algorithm presented here called Low Order Knowledge-Based Algorithm (LOKBA) is capable of solving small to large size problems that might be impossible or too complicated to be solved by other methods. Also, it should be noted that many previous works such as DINS, RINS, and LB are reported for problems of smaller size than the sizes that are tested in this work. Hence, there is a potential for future research to combine such approaches with the algorithm presented here. A very good initial solution within a very short time even for problems of one million variables can be produced with the proposed algorithm and then be improved further by methods such as DINS. Also, combination of exact methods with the algorithm appears to be a possible technique to reduce the problem complexity. For example, the branch & bound’s lower bound could be improved surprisingly if the value of the final solution of this algorithm is used as the lower bound.

2. LOKBA Algorithm

Prior to presenting the steps of the proposed algorithm in the work at hand it is necessary to first define some terms and concepts specific to this algorithm. These terms are explained in section 3.1 and then the algorithm is given in section3.2.

2.1. Definitions

Complement of a variable

The "complement" of a variable x_i as y_i such that

$$x_i + y_i = 1 \quad (1)$$

Complement-constraints

Consider the following BIP sample problem:

Min $z = x_1 - 2x_2 - 9x_3 + 0x_4 + 3x_5 + 0x_6 + 8x_7 - 5x_8$

Subjected to:

$-x_1 - x_2 - x_3 + x_4 - x_5 - x_6 - x_7 - x_8 <= -2$

$-3x_1 + 1x_2 - 2x_3 + 1x_4 + 6x_5 + 0x_6 + 5x_7 - x_8 >= 9$

$+2x_1 + x_2 + 2x_3 + 0x_4 + 9x_5 - 3x_6 - 3x_7 + x_8 >= 7$

For each constraint, a new constraint can be formed using the complement variable. These constraints will be called "complement-constraints" from this point forward. For example, the second constraint in the above sample problem can be transformed to its complement-constraint as follows:

$$-3x_1 + 1x_2 - 2x_3 + 1x_4 + 6x_5 + 0x_6 + 5x_7 - x_8 >= 9$$

&

$$-3(x_1 + y_1) + 1(x_2 + y_2) - 2(x_3 + y_3) + 1(x_4 + y_4) + 6(x_5 + y_5) + 0(x_6 + y_6) + 5(x_7 + y_7) - 1(x_8 + y_8) = 7$$

So the complement constraint will be:

$-3y_1 + 1y_2 - 2y_3 + 1y_4 + 6y_5 + 0y_6 + 5y_7 - y_8 =< -2$

Where $x_i + y_i = 1$.

Superior versus inferior constraints

As shown above, in each pair of a constraint and its complement-constraint, one has greater-equal sign and the other smaller-equal sign. From this point forward, the constraint with greater-equal sign is called superior constraint and represented with a ***C*** symbol. Likewise, the smaller-equal constraint is called inferior constraint and represented with a ***Ø***.

Standard problem

In this work, a problem is defined as "standard problem" when:

1. The objective is maximization.
2. The signs of all constraints and complement-constraints must be either greater-equal or smaller-equal.
3. All factors in the objective function and the constraints are positive integers.

All BIP problems can be transformed into a standard problem. The objective function of a minimization problem is multiplied by -1 to form a maximization problem. For a constraint with greater sign, a value 1 is added to the right hand side (RHS) and the sign is then changed into greater-equal. Likewise, for a constraint with smaller sign, a value 1 is subtracted from the RHS and the sign is then changed into smaller-equal. Wherever a non-integer factor exists for a variable in the objective function or in a constraint, that objective function or that constraint can be multiplied to an appropriate number to satisfy the integer requirement. Also, for variables with negative factor, whether in a constraint or in the objective function, the complement of that variable is used. This is shown in the following standardization example.

In the first step, a maximization problem is formed:

Max **z** = $-x_1+2x_2+9x_3+0x_4-3x_5+0x_6-8x_7+5x_8$

To have positive-sign factors only:

Max **z** = $y_1+2x_2+9x_3+0x_4+3y_5+0x_6+8y_7+5x_8-12$

The equivalent for the objective function will be:

Max **z** = $y_1+2x_2+9x_3+0x_4+3y_5+0x_6+8y_7+5x_8$

The constraints are changed to:

$y_1+y_2+y_3+x_4+y_5+y_6+y_7+y_8 <= 5$

$3y_1+1x_2+2y_3+1x_4+6x_5+0x_6+5x_7+y_8 >= 15$

$2x_1+x_2+2x_3+0x_4+9x_5+3y_6+3y_7+x_8 >= 13$

Now the complement-constraints are formed:

$x_1+x_2+x_3+y_4+x_5+x_6+x_7+x_8 >= 3$

$3x_1+1y_2+2x_3+1y_4+6y_5+0y_6+5y_7+x_8 <= 4$

$2y_1+y_2+2y_3+0y_4+9y_5+3x_6+3x_7+y_8 <= 8$

Finally the standard form of the example is this:

Max **z** = $y_1+2x_2+9x_3+0x_4+3y_5+0x_6+8y_7+5x_8$

Superior constraints:

$x_1+x_2+x_3+y_4+x_5+x_6+x_7+x_8 >= 3$

$3y_1+1x_2+2y_3+1x_4+6x_5+0x_6+5x_7+y_8 >= 15$

$2x_1+x_2+2x_3+0x_4+9x_5+3y_6+3y_7+x_8 >= 13$

Inferior constraints:

$y_1+y_2+y_3+x_4+y_5+y_6+y_7+y_8 <= 5$

$3x_1+1y_2+2x_3+1y_4+6y_5+0y_6+5y_7+x_8 <= 4$

$2y_1+y_2+2y_3+0y_4+9y_5+3x_6+3x_7+y_8 <= 8$

Where $x_i + y_i = 1$.

Weight assignment

Obviously in the above example, the eight variables ($x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8$ or $y_1,y_2, y_3, y_4, y_5, y_6, y_7,y_8$) can take each one of the two 0 or 1 values and therefore the problem has 2^8 = 256 alternatives. Testing all alternatives to find the optimum for even a normal size problem is computationally expensive. In the exact solution approaches such as Branch-

and-Bound (B&B) the variables are prioritized in the ascending order of their factors in the objective function only and their effects in satisfying the constraints are ignored. Therefore, it frequently happens that a variable is assigned as 1 too early although in the optimum solution that variable must be 0. Hence, too many alternatives might be tested to correct the value for this variable in order to find the optimum solution. Furthermore to ensure and verify the optimality, even when the optimum is found in the early steps, still too many alternatives are often tested. To reduce the number of alternatives, in addition to the factors of variables in the objective function, the role of variables in satisfying the constraints has to be considered as well. Lots of effort in the work at hand was dedicated to consider the role of the variables in the constraints and based on the results the following approach is chosen.

Determining the Coefficient of Constraints (CoC)

Each constraint represents a set of acceptable solutions and it seems a valid claim to say that typically a constraint with smaller set has a higher influence on region for the optimum solution and thus it has to have a higher effect coefficient. The solutions for the inferior and superior constraints are corresponding to one another and consequently the numbers of elements in both sets are equal ($n(\boldsymbol{C}) = n(\boldsymbol{\emptyset})$). Using the above explanations, it can be stated that the effect of a constraint and its complement-constraint are equal or in other words each superior constraint and its corresponding inferior constraint have similar influence on the region for the optimum solution.
Because all variables and their complements can have a binary (0 or 1) value, the phrases "$x_1+x_2+x_3+y_4+x_5+x_6+x_7+x_8 >= 3$" or "$y_1+y_2+y_3+x_4+y_5+y_6+y_7+y_8 <= 5$" in the above example will have a value between 0 to 8. The RHS of a superior constraint after standardization is shown by P and the RHS of the inferior constraint after standardization is shown by S. It is obvious that increasing ***P*** or decreasing **S** each will reduce the size of sets ***C*** and ***Ø*** accordingly and consequently it increases the effect of that constraint. Once the ***P*** for all constraints (The vector of ***P***) are non-positive then the problem is fully feasible. On the other hand, if one element of the vector ***S*** is negative, then the problem is infeasible. There will be no such case that the last 2 statements (All ***P***'s non-positive and at least one ***S*** negative) occur simultaneously. This is because ***P***+***S*** is the sum of the factors in the constraints and is non-negative always.

Hence, *CoC* is defined as:

$$CoC = {}^{P}/_{S} \qquad (2)$$

Determining the Effect Coefficient for each variable (SNCoV, SPCoV)

Considering a standard problem, the superior constraints can be written in the general form as:

$$\sum a_j . y_j \geq P \qquad a_j \geq 0 \ \ \& \ \ y_j \in \{0,1\} \tag{3}$$

and the inferior constraints can also be written in the following general form:

$$\sum a_j . x_j \leq S \qquad a_j \geq 0 \ \ \& \ \ x_j \in \{0,1\} \tag{4}$$

Clearly:

$$\sum a_j = P + S \tag{5}$$

On the other hand, the effect of k^{th} variable in satisfying the constraints can be represented by $a_k / \sum a_j$ (Note that all variables are binary) which is the same as

$$a_k / \mathbf{P + S}.$$

Variables can take a value 0 or 1, each value results in a different effect on the constraints. To analyze the amount of influence of each variable in the constraints, two new coefficients are defined here. In a standard problem considering all variables in the objective function, *SPCoV* is defined below such that it illustrates the effect of each variable once given a value 1. Similarly, *SNCoV* is defined such that it illustrates the effects of each variable once given a value 0. Thus, having *m* constraints and *n* variables these coefficients are defined as:

$$SPCoV(x_k) = \sum_{i=1}^{m} \left(\frac{a_{ik}}{P_i + S_i} \times \frac{P_i}{S_i} \right) \qquad a_{ik} \geq 0 \ \ \& \ \ \forall 1 \leq k \leq n \tag{6}$$

$$SNCoV(x_k) = \sum_{i=1}^{m} \left(\frac{a_{ik}}{P_i + S_i} \times \frac{P_i}{S_i} \right) \qquad a_{ik} \leq 0 \ \ \& \ \ \forall 1 \leq k \leq n \tag{7}$$

Note that:

$$SNCoV(x_k) = \sum_{i=1}^{m} \left(\frac{a_{lj}}{P_l + S_l} \times \frac{P_l}{S_l} \right) = P_l / S_l = CoC(C_l) = CoC(\emptyset_l) \qquad \forall 1 \leq l \leq m \tag{8}$$

Normalization of objective function factors

To normalize the objective function factors, their absolute value of each is divided to the sum of all their absolute values. Evidently, the total of all normalized factors equals one.

Normalization of *SPCoV* and *SNCoV* coefficients

Although usually to normalize a series of values they are divided to their total such that the sum of normalized values is 1, here another approach is used. In the solution

International Journal of Academic Research in Business and Social Sciences
January 2012, Vol. 2, No. 1
ISSN: 2222-6990

algorithm presented in this work, the *SPCoV* and *SNCoV* coefficients relate to the feasibility of the problem since they are formed based on the constraints. Thus, when in a problem the importance of feasibility is higher than its optimality, these feasibility coefficients (i.e., all *SPCoV*'s and *SNCoV*'s) have to have higher effect than the objective function factors. This means when the difficulty of the constraints increases (the feasibility will reduce and therefore its importance will increase), then the sum of all feasibility coefficients must also increase in comparison to the normalized factors of the objective function. For this purpose, to normalize the feasibility coefficients (Instead of dividing to their total), they are divided to the number of constraints (*m*). The result can be larger than one, whereas the sum of normalized objective function factors is always one.

Difficulty coefficient of the sum of all constraints = $\sum_{j=1}^{n} \frac{(SNCoV_j + SPCoV_j)}{m}$ (9)

Determining Final Coefficient (*FCo*) for each variable

It should be noted that using constraints alone to prioritize variables in taking value does not provide a comprehensive solution strategy. This is similar to considering the ascending order of the objective function factors for variables in the Branch and Bound approach that sometimes leads to a prolonged or too extensive search. Therefore, a more reliable approach is to use both the factors of the constraints and the factors of the objective function. Prior to explaining how to use both of these factors, it should be pointed out here that normally the two following approaches exist for solving binary problems:

1- The general approach considers both the feasibility and the optimality of the solution. Here it is tried to find the feasible solution close to optimum as much as possible.

2- The particular approach considers the feasibility of the solution only. This approach is used when the solution space is significantly small and finding a feasible solution is vital. Here the probability of finding a feasible solution is greater compared to the general approach as the focus is on feasibility only. Obviously an optimum but non-feasible solution is of no use. Hence, in the algorithm presented in this work the general approach is employed first and after several iterations to get closer to optimum the solution space shrinks and becomes smaller. Next whenever finding a better feasible solution through general approach is not possible, the particular approach is employed.
In the general approach both of the above mentioned factors (constraints and objective function) are used simultaneously in the manner that follows. Once a variable is 1, then clearly the objective function increment is the value of its factor in that function. Furthermore, the constraints are satisfied in the amount of its *SPCoV*. On the other hand, once a variable is 0, the objective function does not change and the constraints are then satisfied in the amount of *SNCoV*. Therefore, a new coefficient can be defined

International Journal of Academic Research in Business and Social Sciences
January 2012, Vol. 2, No. 1
ISSN: 2222-6990

for each variable by adding *SPCoV* to the normalized factor of that variable in the objective function and then dividing them to *SNCoV*.

$$\text{FCo}(x_j) = \frac{\text{SPCoV}(x_j) + \text{the normalized factor of variable } x_j \text{ in the objective function}}{\text{SNCoV}(x_j)} \quad (10)$$

In the particular approach the objective function factors are neglected and therefore *FCo* is calculated as below:

$$\text{FCo}\,(x_j) = \frac{\text{SPCoV}(x_j)}{\text{SNCoV}(x_j)} \quad (11)$$

Once a variable has an *FCo* greater than 1 it has a high merit to take the value 1 with the variable with highest *FCo* having the highest merit. On the other hand a variable with *FCo* smaller-equal 1 has a high merit to become 0 with the variable with the smallest *FCo* having the highest merit to become zero. The details of how to use *FCo* will be explained later in the proposed algorithm.

Strong variable

In the Branch and Bound approach many branches and sub-branches are created and then while using the feasibility test, it can be observed that some of the variables have to be fixed to a special value. In the proposed algorithm in this work, this fact is discovered earlier in higher level branches and the feasibility tests in lower levels are skipped. This is done through defining strong variables. In the proposed work during the iterations of the algorithm variables are gradually assigned 0 or 1 values and consequently the RHS of the constraints are decreased accordingly. Whenever in an inferior constraint the factor of a variable is greater than the RHS of that constraint, then that variable is considered strong and is assigned to zero. This is because inferior constraints have smaller-equal sign and all variables have non-negative factors in a standard problem. Therefore, if the factor of a variable is greater than the RHS and if assigned to 1, then obviously the result of multiplying the factor to 1 violates the constraint.
The assigned value for the strong variables is fixed contrary to the other variables that take a value based on the weighting scheme.

Firm variables

Once the vector of factors for a variables in all of the superior constraints of a standard problem is greater-equal zero, then *SNCoV*=0 and that variable is firm and will be assigned to 1. This definition for firm variable is once the general approach is employed. For the particular approach in addition to those firm variables with *SNCoV*=0 (which are

assigned to 1) also the variables with *SPCoV*=0 are considered as firm which will be assigned to 0.

Obsolete constraints

As mentioned earlier, in the proposed algorithm variables (like strong or firm) are gradually assigned values during the iterations of the algorithm. This value for each variable is multiplied to its factor in each constraint and then deducted from the RHS. And at some point the RHS of the superior constraints will become zero or negative. A superior constraint has greater-equal sign and a standard problem has non-negative factors only. Thus, a zero or negative RHS means that the remaining variables can take any value and the constraint will be true anyways. Hence, this constraint and its complement become obsolete.

2.2. The algorithm steps

The algorithm works on the concept of finding a solution and if possible improving this solution further by adding extra constraints. To find a solution in each step, knowledge is extracted from the values of the factors in the objective function and the constraints. This knowledge is then processed further in a loop called "improvement loop" to create smaller problems intelligently. Next the knowledge is also used in a different manner to assign weights to the constraints and the variables. Based on the weighting scheme, the variables are assigned values. From the assigned values, a new knowledge is extracted. This new knowledge upgrades the previous knowledge and the algorithm uses this as a feedback for future improvement. The algorithm stops with the best solution found so far once there can be no further improvements.

The procedure of the algorithm is such that it produces a better solution in each of the iterations while using all of the knowledge that can be extracted from the problem factors to improve the solutions. Therefore, it can be expected from the algorithm to produce promising results. This is proven by the results that are discussed later.

The algorithm is described in the following steps and the flowchart of the algorithm is presented in figure 1. For the example of each step, please refer to the comprehensive example given at the end of this paper in the appendix. The algorithm has an initialization step where it is transformed into a standard problem. The solution approach can also be explained using a table. The table is introduced for the simplicity and better understanding of the algorithm. Consider a BIP problem with n variables and m constraints.

1. The objective function has to be of maximization type (Max-type). So if the objective of BIP is minimization, multiply the objective to -1. Then, the first *n* elements in the first row of the table are the signs of the objective function factors, each is written in one column (from column 1 to *n*) accordingly.
2. The second row of the table is devoted to variables. This means that the second row is filled with *x*'s and *y*'s with index 1 for the first column, index 2 for the second column, index 3 for third column and so on till the n^{th} column. In each column if the first row has a positive sign, then variable x_j is used and otherwise

(negative sign) y_j is used where j = 1 to n. It should be noted that this way of filling the table is towards the standardization of the problem.

3. The third row of the table (from column 1 to column n) is filled with the absolute values of the objective function factors.
4. Constraints will be written from the 4^{th} row onwards with one row for each constraint in a manner that will follow (Up to now there are m+3 rows). First all of the constraints have to be transformed to greater-equal or smaller-equal type by the technique explained in standardization of the problem. For each constraint of the problem before standardization that have greater-equal sign, a positive sign must be written in the n+1 column of the row specific to that constraint. Similarly, for smaller-equal constraints a negative sign is written. The factors of variables in each constraint will then be multiplied to this sign (written in $(n+1)^{th}$ column) and also multiplied to the sign written in the first row of the table of the column.
5. According to that variable. The resulting values will be written in the row for that constraint from column 1 to n.
6. In the same row for a constraint, ***P*** is written in the column n+2 and ***S*** is written in column n+3. ***P*** and ***S*** are calculated as below.

When the sign in column n+1 is positive, then:

P = RHS – (The sum of negative factors in the constraints prior to standardization) (12)

S = (The sum of positive factors in the constraints prior to standardization) – RHS (13)

When the sign in column n+1 is negative, then:

P = (The sum of positive factors in the constraints prior to standardization) – RHS (14)

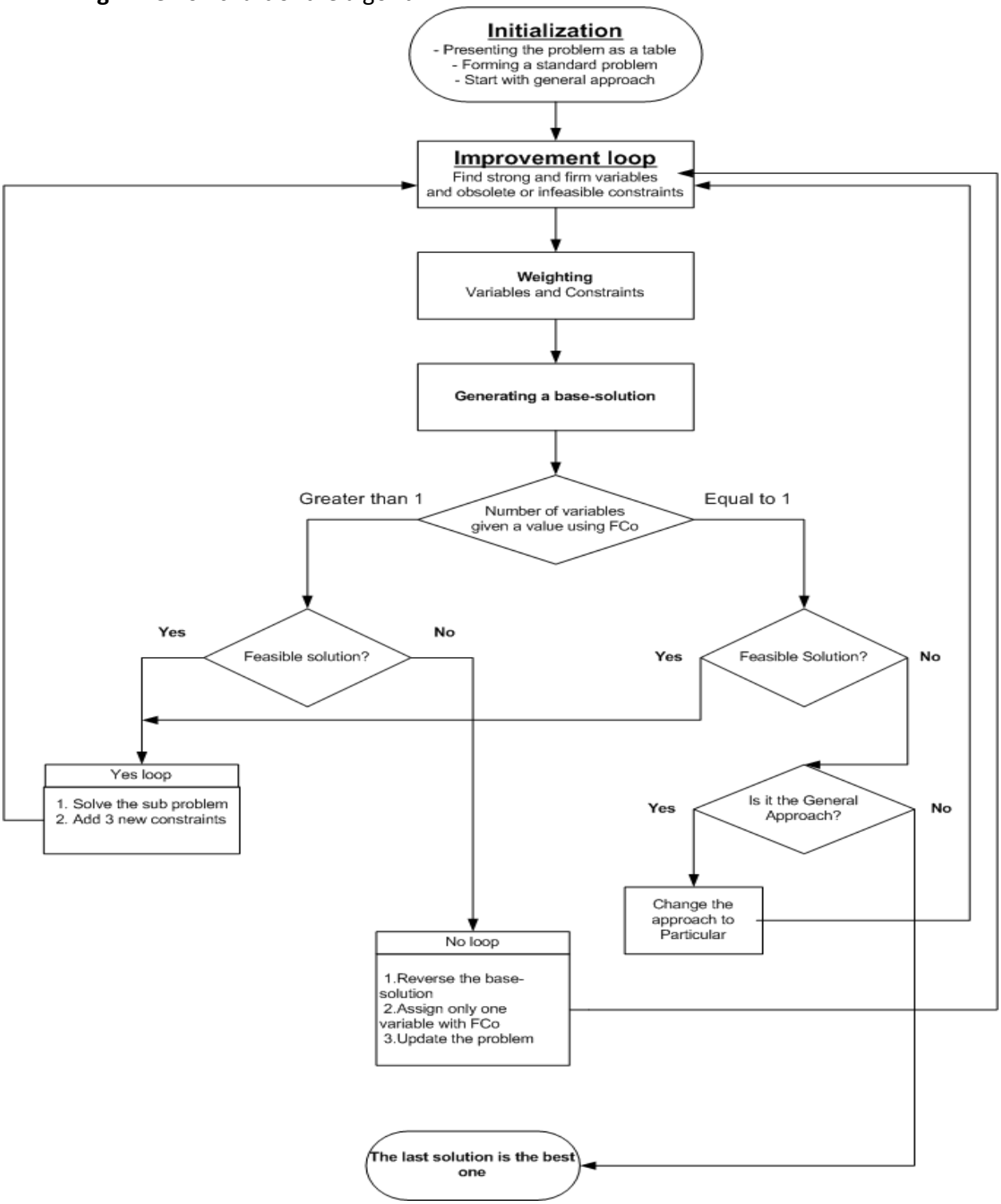
S = RHS – (The sum of negative factors in the constraints prior to standardization) (15)

It should be noted that:

P+**S** = Sum of the absolute values of the factors of the constraint (16)

The primary table, which basically is the initialization section of the algorithm, is now ready. Also consider the general approach.

Fig1 The flowchart of the algorithm



7. Now the improvement loop:
   - If there is a constraint with negative *S*, then go to step 18.

- Find the strong and firm variables and the obsolete constraints.
- Assign the values of the strong and firm variables and omit the obsolete constraints (if any exist)
- Update the ***P*** and ***S*** values for the new problem (Note that the rest of the table does not change). The updating is in this form:
  When a variable is assigned to 1, the positive factors are deducted from its *P* and the negative factors are added to its *S*. When a variable is assigned to 0, the positive factors are deducted from its *S* and the negative factors are added to its *P*.
- Loop back to find again the strong and firm variables and obsolete constraints until there are no more such variables and constraints. It is needed to loop back because after each iteration of this loop the new setting of the problem might result in creating the possibility of finding further strong or firm variables and obsolete constraints. Thus, once the loop ends it is certain that all the leftover variables and constraints need another approach for being evaluated. This is done through the weighting scheme below. All the values given to the specified variables in this step must be written in row *m*+10. The values for remaining variables (empty columns of this row) will be filled in step 12 after weighting steps.

8. Beginning of the weighting scheme starts here with calculating the *CoC* of each constraint divided to (*P*+*S*) of that constraint and writing the result in $(n+4)^{th}$ column in the row for that constraint.
9. In this step the row *m*+4 will be filled from column 1 to *n*. For each variable (i.e., for each column) the sum of the product of the positive factors written in step 4 for that variable and the number calculated in the previous step for each constraint will be written in row *m*+4 in the same column. This value is *SPCoV* of the variable according to that column. Similarly, the sum of the product of the negative factors written in step 4 for each variable and the number calculated in the previous step for each constraint will be written in row *m*+5 in the same column. This is *SNCoV* of the associated variable.
10. Rows *m*+6 and *m*+7 are the normalized values of the two previous rows accordingly.
11. Row *m*+8 is the normalized values of the objective function factors (row 3).
12. *FCo* is now calculated for variables that have not been given any value up to now and written in row *m*+9. Weighting is now completed.
13. Using the *FCo* weights of the previous step, the associated variables are assigned 0 or 1 value according to the criteria that those with *FCo* greater than 1 are given value 1 and those smaller-equal 1 are given the value 0. This is a solution created for the BIP problem and is called the "base solution" hereafter and this base solution will fill the blank elements of row *m*+10. If only one variable is given a value in this step, then update the problem (***P*** and ***S***) similar to step 6 and go to step 17. Otherwise continue.
14. The feasibility of the solution is tested at this step. If yes, go to next step (Yes-Loop). Otherwise go to step 16 (No-Loop).

International Journal of Academic Research in Business and Social Sciences
January 2012, Vol. 2, No. 1
ISSN: 2222-6990

15. Yes-Loop is to improve the found feasible solution as much as possible. In this loop we put aside all those variables with value 1 and update the problem using these values. The approach of updating is the same as explained in step 6. Therefore, the number of variables of this sub-problem is the number of variables with value zero prior to this step (It has less than *n* variables). All the variables with value 1 are omitted from the objective function to form the new objective function. Also an additional constraint is added to the sub-problem. This constraint is as follows:

    New objective function ≥ The maximum of {1 & the factor of the new objective function with minimum value} (17)

    This sub-problem is now solved by going to the step 6.
    The yes-loop continues till the last sub-problem is infeasible.
16. Now that the solution has been improved. Note that the improved solution has *n* variables. The objective function value is then calculated for this improved solution and a new problem is formed by adding this constraint:

    Objective function ≥ 1+ Objective function value for the improved solution (18)

    To escape from local optimum the two following constraints are also added that forces to find a new solution if possible.

    The sum of all variables that had value 1 ≤ (the number of these variables -1) (19)

    The sum of all variables that had value 0 ≥ 1 (20)

    With this new problem go to step 6. This forces the quest for better solution to continue till at some level, the algorithm stops due to infeasibility.
17. No-Loop starts here. First reverse the step 12 as if it just past step 11. Find the max and min of *FCo*'s calculated in step 11. If the product of this max and min is greater than 1 then assign the value 1 to the variable that has this max *FCo*. Otherwise, assign 0 to the variable with min *FCo*. Update ***P*** and ***S*** in the same way that was stated in step 6. Take this new problem and go to step 6.
18. Feasibility test. If feasible, go to step 14. Otherwise continue.
19. Check to see whether the general or the particular approach is used. If the approach is general currently, then change to particular approach and go to step 6. Otherwise (particular approach), the last found solution is the final solution.

3. Results

The algorithm is tested in various manners using a normal PC. First, the updated MIPLIB 3.0 which is the famous Mixed Integer Programming Library is used (http://miplib.zib.de/miplib2003.php). To compare LOKBA with other recent Algorithms (Default Cplex, LB, RINS, DINS, VNDS and VNSB), only the results of problems solved by

International Journal of Academic Research in Business and Social Sciences
January 2012, Vol. 2, No. 1
ISSN: 2222-6990

other algorithms are reported here. Table 1 illustrates the results of LOKBA in column 6 for the eight problems given in the first column. The results of other algorithms (Default Cplex, LB, RINS and DINS) reported in (Ghosh, 2007) are shown in columns 2 to 5 for comparison. The run time for LOKBA on a normal PC (2.0 GHz Intel) is given in column 7. It should be noted that the run time for all other algorithms for all eight problems is one CPU-hour on a 2403 MHz AMD Athlon processor. The speed of LOKBA is much more with results not too far from the others.

Table 1: Optimality and running time comparison

Problem Name	*Default Cplex*	*LB*	*RINS*	*DINS*	*LOKBA*	*LOKBA Run Time (Sec)*
seymour	0.995	0.995	1.000	0.998	1.000	107.096
sp97ar	0.996	0.995	0.997	1.000	0.936	246.563
sp97ic	0.992	0.994	0.994	1.000	0.943	215.291
sp98ar	0.998	0.999	0.998	0.998	0.964	110.834
sp98ic	0.997	0.999	0.998	0.999	0.957	133.480
fast0507	1.000	0.994	0.994	1.000	0.992	1476.107
harp2	1.000	1.000	1.000	1.000	1.000	75.329
t1717	0.921	0.981	0.940	0.921	0.952	1733.008
ds	0.888	0.857	0.888	0.939	0.843	1558.748

To further verify the algorithm and compare with other recent works, the optimality and running time of LOKBA is compared to VNSB and VNDS that are two recent methods for 0-1 programming. Table 2 reports the results of the algorithms and their run times in seconds. It also illustrates that LOKBA has great speed with very good solutions.

Table 2: Optimality and running time comparison

Problem Name	***VNDS***	***VNDS Run Time (Sec)***	***VNSB***	***VNSB Run Time (Sec)***	***LOKBA***	***LOKBA Run Time (Sec)***
seymour	0.995	9151	1.000	15995	1.000	107.096
sp97ar	1.000	16933	0.999	5614	0.986	246.563
sp97ic	0.991	2014	1.000	7844	0.993	215.291
sp98ar	0.999	7173	1.000	6337	0.994	110.834
sp98ic	1.000	2724	1.000	4993	0.997	133.480

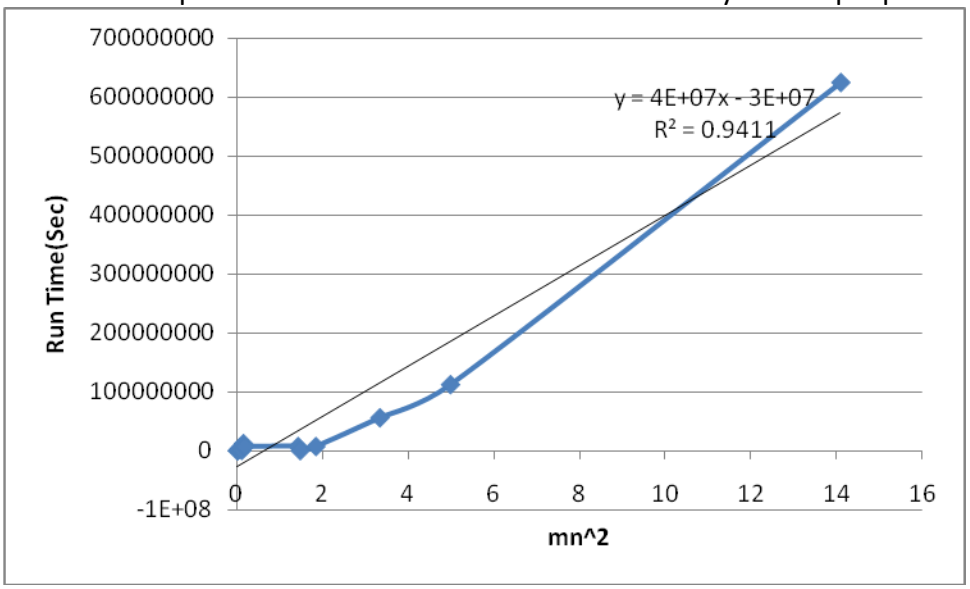
The LOKBA algorithm is also tested using some sample problems taken from the OR library website (http://people.brunel.ac.uk/~mastjjb/jeb/info.html). The results are shown in table 3. The size of these sample problems were ranging from problems with 100 variables and 15 constraints to problems with 2500 variables and 100 constraints. The run times of the algorithm range from 0.01 to 14 seconds. The maximum deviation from the given solution was 1.3 percent and the average deviation was 0.4 percent.

Table 3: The results for 10 problems from OR library

n-m	*Optimum*	*LOKBA*	*Run Time*	*Ratio*
100-15	3766	3766	0.011	1.000
150-25	5650	5650	1.479	1.000
150-50	5764	5693	0.024	0.988
200-25	7557	7494	0.114	0.992
200-50	7672	7571	0.034	0.987
500-25	19215	19215	0.089	1.000
500-50	18801	18671	0.152	0.993
1500-25	58085	58085	3.339	1.000
1500-50	57292	57110	4.987	0.997
2500-100	95231	94896	14.096	0.996

In order to analyze the sensitivity of the algorithm to the increase of the number of variables and constraints a graph is shown in Figure 2. After applying regression method on the results shown in table 3, the run time (which represents algorithm's order) can be approximated as a function of the number of constraints multiplied to the square of the number of variables (mn^2). The approximation using the correlation coefficient is R^2 = 94.11 %.

Fig 2: The relationship between run time and mn^2 for OR library 10 sample problems



Next, as another way of testing the proposed algorithm, the results of the algorithm by Fortin and Tseveendorj (Fortin and Tseveendorj, 2009) are also used. All of their sample problems have 500 variables and 30 constraints. They have not provided the run times of their approach. The deviation of LOKBA from their results is 1.5 percent with run times of about 1 second.

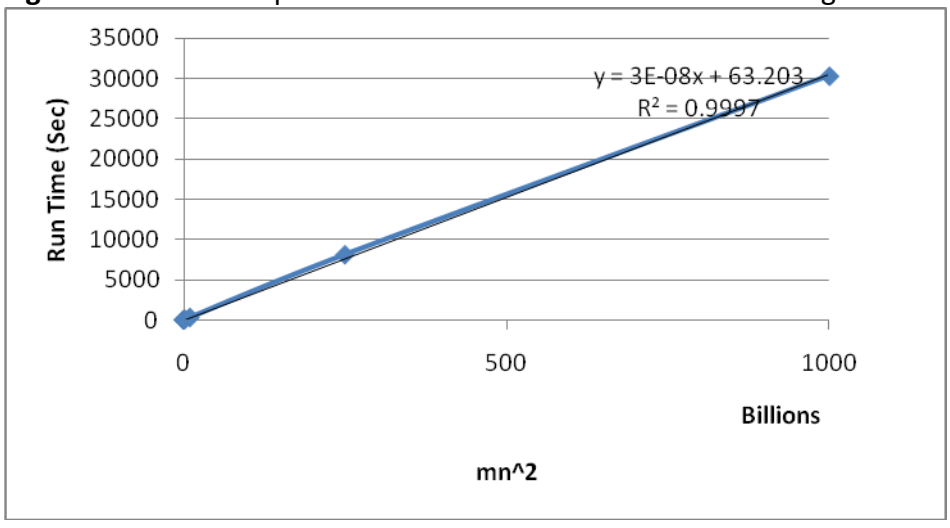
In addition, several BIP problems of small to large size (up to 1,000,000 variables) with known optimal solutions are generated and tested. The algorithm produces the optimal solution in all of these cases. The results are shown in table 4. The run times are again surprisingly low. The running time for the problem with 1,000,000 variables was less than 8.5 hours; for the problem with 10,000 variables was less than 4 seconds and for smaller problems was even a fraction of a second.

Table 4: The results for 5 large size instances

n-m	*Optimum solution*	*Solution by the algorithm*	*LOKBA Run Time (Sec)*
10000-1	9500500	9500500	3.434
50000-1	49500500	49500500	63.674
100000-1	99500500	99500500	297.571
500000-1	499500500	499500500	8104.330
1000000-1	1998001000	1998001000	30337.295

Although, in this step many different BIP problems were tested, a set of problems with similar structures were used in order to analyze the sensitivity of the algorithm when the number of variables is growing. Using the data of table 4, figure 3 demonstrates similar behavior shown in figure 2 with an even better approximation. Hence, it is predictable that the relationship will be the same for any other binary problems. Therefore, the low sensitivity of the presented approach to the size of the problem in comparison to the other approaches make this approach suitable for solving extra-large problems.

Fig 3: The relationship between run time and mn^2 for above large size samples



4. Conclusion and future works

There are definitely some information and knowledge embedded in the factors of the constraints and the objective function of any binary integer programming problem. This

knowledge, if extracted and implemented intelligently, can help in finding the optimal solution. In this work, this is attempted and a novel algorithm is developed that uses this knowledge in many different levels. The knowledge is employed to form a so-called novel "standard" problem and then the variables and constraints are then classified in an innovative manner. Next the constraints and variables are weighted. The solutions are improved towards optimum in several iterations. In these iterations the variables are assigned values and new knowledge is created again that finally results in an optimal or near optimal solution within a short amount of time. The efficiency of the algorithm is tested using many different BIP problems. The algorithm produces optimal or near optimal solutions in all test cases.

Due to the optimality of the numerous results acquired in amazingly short run times, the algorithm is promising. It has the ability to solve any type of BIP problems (whether they encompass either positive or negative factors or both of them and without any limitation for the number of constraints) with low sensitivity to the size of the problem. Based on these characteristics, there is high potential for extending this approach by combining it with other methods. For example, it can be combined with many previous works such as DINS, RINS, LB, VNSB and VNDS that are used for problems with much less variables in comparison to the problems tested in this work. The proposed algorithm can provide an initial solution for a very large problem with satisfactory gap from the optimum in a short time. Then, the other methods can improve the solution and decrease the gap. However, it should be noted that a solution with some gap for a real world huge commercial problems which normally have a considerable amount of uncertainty might be satisfactory. Instead of very exact solutions, there might be a need to run the BIP models frequently in a short time. Another possibility is to consider fuzziness and uncertainty of the data in the algorithm.

5. Acknowledgement

The authors are grateful for the financial support by the University of Tehran, College of Engineering, for this research with grant number 27768/1/01.

References

Baeck, Fogel D, and Michalewicz Z (1997). Handbook of Evolutionary Computation, Oxf. Univ. Press, NY.

Captivo ME, Climaco J, Figueira J, Martins E, Santos JL (2003),Solving Bicriteria 0-1 Knapsack Problems Using a Labeling Algorithm, Comput. Oper. Res.,30: 1865-1886

Chu PC and, Beasley JE (1998), a Genetic Algorithm for the Multidimensional Knapsack Problem, J. Heuristics, 4: 63-86

Danna E, Rothberg E, and Pape CL (2005), Exploring Relaxation Induced Neighborhoods to Improve MIP Solutions. Math. Program., 102:71-90

Domitrescu I and Stuetzle T (2003), Combinations of Local Search and Exact Algorithms, Applications of Evolutionary Computations, Lect. Notes. Comput. Sc., 2611:211-223, Springer

Fischetti M and Lodi A (2003), Local Branching, Math. Program. B, 98: 23-49

Fortin D, Tseveendorj I (2009), a Trust Branching Path for Zero-One Programming, Eur. J. Oper. Res., 197: 439-445

Ghosh S (2007), DINS, a MIP Improvement Heuristic, Lect. Notes Comput. Sc., 4513:310-323, Springer

Glover F and Laguna M (1997), Tabu Search, Klu. Acad. Publ.

Glover F, Laguna M, and Marti R (2000), Fundamentals of Scatter Search and Path Relinking, Control. Cybern., 39(3): 653 681

Haartman KV, Kohn W, Zabinsky ZB (2008), A Meta-Control Algorithm for Generating Approximate Solutions to Binary Integer Programming Problems, Nonlinear. Anal. Hybrid Syst., 2:1232-1244

Hansen P, Mladenovic N, Urosevic D (2006),Variable Neighborhood Search and Local Branching, Comput. Oper. Res., 33: 3034-3045

Kellerer H, Pferschy U, and Pisinger D (2004): Knapsack Problems, Springer

Kirkpatrick S, Gellat C and Vecchi M (1983), Optimization by Simulated Annealing, Science, 220:671-680

Larranaga P and Lozano J (2001), Estimation of Distribution Algorithms, a New Tool for Evolutionary Computation, Klu. Acad. Publ.

Lazic J, Hanafi S, Mladenovic N, Urosevic D (2009), Variable Neighborhood Decomposition Search for 0-1 Mixed Integer Programs, Comput. Oper. Res.

P. Moscato and C. Cotta, A Gentle Introduction to Memetic Algorithms, in Handbook of Metaheuristics, Chapter 5, pp.105-144, F. Glover and G. Kochenberger (Eds.), Klu. Acad. Publ., Boston, Massachusetts, USA, (2003), ISBN:1-4020-7263-5

Namazifar M, Miller AJ (2008), a Parallel Macro Partitioning Framework for Solving Mixed Integer Programs, Lect. Notes Comput. Sc., 5015: 343-348, Springer

Nemhauser G and Wolsey L (1988), Integer and Combinatorial Optimization, J. Wiley , NY.

Vasquez M , Hao JK (2001), a Hybrid Approach for the 0-1 Multidimensional Knapsack problem, Int. Joint. Conf. Artif.,

International Journal of Academic Research in Business and Social Sciences
January 2012, Vol. 2, No. 1
ISSN: 2222-6990

Wang Z, Xing W (2009), a Successive Approximation Algorithm for the Multiple Knapsack Problem, J. Comb. Optim.,17: 347-366
Wolsey LA (1998), Integer Programming, J. Wiley, NY.