Optimal Boolean Programming with Graphs

Benedek Nagy

Eastern Mediterranean University, Department of Mathematics, Famagusta, North Cyprus, via Mersin-10, Turkey

Abstract: In this paper, we solve some special types of linear and non-linear Boolean programming problems. We present a method for transforming the used linear 0-1 *inequalities into a weighted directed graph. Allowing equalities our conditions are non-linear, but the transformation to weighted directed graphs works also in these cases. In graph representations, the "critical edges" are used to represent the non-linear conditions. Basic, modified and extended Boolean programming problems are investigated. Linear goal-functions are used in optimization. The presented algorithm, similarly as algorithms for knapsack-problems, gives a relatively good solution, moreover, the algorithm extended by the backtrack graph-search strategy, guarantees optimal solutions for the considered problems.*

Keywords: Boolean-programming; 0-1 *inequalities; optimization; knapsack problem; greedy algorithm; backtracking; graphs*

1 Introduction

There are some special integer-valued programming problems in which the values of variables must be in the set {0,1}. These Boolean programming (BP) problems are well known in the literature. We refer to [7] as a classical textbook on the topic related to optimization and Boolean programming. BP problems are ubiquitous in Artificial Intelligence. Planning with resource constraints, satisfiability testing and winner determination in combinatorial auctions are all belonging to this type of problems [18, 21]. The knapsack problems [9, 19, 22] are also in this class. In many logical problems, e.g., in some logic puzzles, the same or nearly similar conditions occur [10-17, 20]. In [3] special redundancy criteria were used to produce minimal number of extended clauses for transforming problems to equivalent ones.

In this paper, we continue the work that has been started in [11-14]. The conditions of a basic BP problem have a translation to a set of linear-programming conditions such that in each condition two variables occur. This is a key issue for representing such problems with graphs. The conditions of a modified BP problem can also be translated to linear conditions, however these conditions, in general,

cannot be written as a set of conditions such that each condition contains only two variables. Despite this, we show how these conditions can still be represented by graphs. In the extended BP problems, the conditions are more general than in the basic or modified problems. Graph-representation [2] of the conditions will be used, where the so-called critical-edges refer to the conditions where some of them are considered together in the linear programming model in cases of modified and extended BP problems. There is a goal-function to do optimization. We search the possible solution(s) under the criteria that the goal-function has maximal (or minimal) value. Based on our graph representation we use local steps (comparing variables) to infer more knowledge about their relations. The algorithm is extended with a greedy and with a backtracking part to ensure to obtain the solution(s). We use a greedy algorithm to find a reasonable solution and, if it is not enough, backtracking to find a better, or even, the best solution.

Our algorithm is based on the algorithms of [12-14]. The algorithm uses the graph representation. It can modify the graph of conditions in such a way that the solutions for the new graph are exactly the same as for the original problem. Using the fact that the value of every variable must be in {0,1} and using some other observations, we can conclude a unique possible value for some variables. Consequently, the steps of the algorithm are the graph modifications and assigning values to variables. We solve the mentioned types of BP problems by our algorithm.

2 Conditions of BP

In this research, we solve problems where each variable can have a value of the set {0, 1}. Now we give formal definitions of various types of BP problems we are working with. Capital letters (sometimes with indices) are used to denote the variables of the problem.

First, the definitions of basic and modified BP problems are given. Then, we define the general case which are called extended BP problems and, in fact, they are generalizations of the modified Boolean programming problems.

In this paper, we will solve the generalized problem, and we will show that the basic and modified problems are its special subcases.

Definition 1. (Basic-, Modified- and Extended BP problem) Let B_i denote the variables of the problem for $i \leq n$ where n denotes the number of variables. If every condition has the form

(1) $B_i \leq B_{r1} \cdot B_{r2} \cdot ... \cdot B_{rm} \cdot (1-B_{p1}) \cdot (1-B_{p2}) \cdot ... \cdot (1-B_{pk})$

then our conditions are of basic BP type.

If every condition is written in one of the following forms

(1) $B_i \le B_{r1} \cdot B_{r2} \cdot ... \cdot B_{rm} \cdot (1-B_{p1}) \cdot (1-B_{p2}) \cdot ... \cdot (1-B_{pk})$

(2) $B_i = B_{r1} \cdot B_{r2} \cdot ... \cdot B_{rm} \cdot (1-B_{p1}) \cdot (1-B_{p2}) \cdot ... \cdot (1-B_{pk})$

and every B_i appears on the left-hand side of at most one condition of type (2), then the conditions of the problem are of modified BP type.
If each condition has one of the following four forms:

(1) $B_i \le B_{r1} \cdot B_{r2} \cdot ... \cdot B_{rm} \cdot (1-B_{p1}) \cdot (1-B_{p2}) \cdot ... \cdot (1-B_{pk})$

(2) $B_i = B_{r1} \cdot B_{r2} \cdot ... \cdot B_{rm} \cdot (1-B_{p1}) \cdot (1-B_{p2}) \cdot ... \cdot (1-B_{pk})$

(3a) $B_i < B_{r1} \cdot B_{r2} \cdot ... \cdot B_{rm} \cdot (1-B_{p1}) \cdot (1-B_{p2}) \cdot ... \cdot (1-B_{pk})$

(3b) $1 - B_i < B_{r1} \cdot B_{r2} \cdot ... \cdot B_{rm} \cdot (1-B_{p1}) \cdot (1-B_{p2}) \cdot ... \cdot (1-B_{pk})$

and every B_i appears on the left-hand side of at most one condition of type (2), then, these conditions are of extended Boolean programming type.
Let our goal-function be in the form

(4) $Z = a_1 \cdot B_1 + a_2 \cdot B_2 + ... + a_n \cdot B_n$, where the values a_i are fixed real numbers.

If our goal is to minimize or maximize the function Z and our conditions are of type basic, modified or extended Boolean programming, then our problem is basic, modified or extended Boolean problem (BP), respectively. ■

We have no restriction on the occurrences of the variables in basic BP and we have only one restriction in case of modified and extended BP. While the main difference between the basic and the modified BP is the possibility of equations, the difference between the modified and the extended BP is the possibility of strict inequalities. Sometimes it happens that our conditions are not in the form as we have defined above, but we can use graph representation and we can still use our method. It may occur in the case when there are more than one conditions of type (2) with the same variable on the left hand side. If we have special type (2) conditions such that only one variable appears in both sides, then we may interchange the variables on the left and on the right hand side, it may help to transform our conditions to the defined form. If some of our conditions look like (3a) but in the left hand side there is not only one variable, but the sum of more than one variables, we can separate the condition to more than one conditions of type (3a) in which the left hand side contains only one variable and the right hand side is the same as in the original condition. If some of our conditions look like (3b) but in the left hand side there is not only one variable, but the number of the variables minus the sum of these variables, then we can write several conditions of type (3a) such that in each of them the left hand side contains exactly one variable and the right hand side is the same as in the original condition.

All basic BP problems have possible solutions, for example, each $B_i = 0$ is a trivial possible solution. Opposite to this, there are some modified and extended BP

conditions which have no solution. A simple example is as follows. Let the only one variable be B. Let our unique condition be B = 1−B. It is easy to see that there is no possible solution in the set {0,1}.

Now, we will define the basis of our graph theoretic approach. We will use directed graphs to represent the conditions of a BP problem, to do this, we need the conditions to be written as a set of conditions containing at most two variables.

Lemma 1. We can write the type (1) conditions of a BP-problem in the form:

(5a) $B_i \le B_{rj}$
(5b) $B_i \le 1-B_{pl}$

Proof. If $B_i = 0$, then the conditions trivially hold in both in the original and in the stated cases. If $B_i = 1$, then at each condition the equality holds. ■

The conditions in any of the forms (5a) and (5b) are called *atomic conditions*. Observe that they are, in fact, linear. Now we show that the conditions of not only the basic type, but the modified and extended BP problems can also be written in linear programming form. The corresponding type (5a) and (5b) conditions are also satisfied for a type (2) condition, but using these new forms, the new conditions are not equivalent to the original condition in form (2).

Lemma 2. The conditions (2), (3a) and (3b) can be written in linear form. A condition of type (2) can be written as the set of corresponding conditions in the form (5a) and (5b) and an additional condition

(2*) $1-B_i \le (1-B_{r1}) + (1-B_{r2}) + ... + (1-B_{rm}) + B_{p1} + B_{p2} + ... + B_{pk}$

Moreover, conditions of the form (3a) can be written as a set of conditions

(3a*) $B_i \le 0$ $\qquad 1 \le B_{rj}$ $\qquad 1 \le 1-B_{pl}$

while conditions of the form (3b) are equivalent to a set of conditions

(3b*) $1 \le B_i$ $\qquad 1 \le B_{rj}$ $\qquad 1 \le 1-B_{pl}$

Proof. Let a type (2) condition be given. Then, in case $B_i = 1$, all corresponding type (5a) and (5b) conditions are equalities and (2*) does not mean any further restrictions. In case $B_i = 0$, however, (2*) takes care about the equality of (2) by forcing either at least one B_{rj} to be 0 or at least one B_{pl} to have value 1. Considering type (3a) or (3b) conditions, the strict inequality hold only if the left hand side has value 0 and the right hand side contains a product of only 1's. Moreover, all the formulae of (3a*) and (3b*) are representing equalities. ■

In the graph we represent the atomic conditions. We will use weighted arrows as edges in the graph according to the types of the conditions. We use abbreviations LHS and RHS for left hand side and right hand side, respectively. We call a relation critical if it is based on an original type (2) relation. It is easy to show that if the variable of the LHS of a type (2) condition has value 1, then the conditions

in forms (5) are equivalent to the original condition. However, in case the value of the variable on the LHS is 0, the conditions are not the same, we must pay attention that at least one element of the product in the RHS must be 0. The critical edges are used to represent these kinds of possible equalities.

Definition 2. (Critical condition) If an atomic condition may represent an equality of type 0 = 0, which comes directly from a type (2) condition of the BP problem, then this atomic condition is critical. ■

We use various weights (labels) to represent the possible conditions between any two variables:
If the value is not a multiplier of 3, then it means that the condition (5a) is true for these variables. If the value is not less than 3, then it means that the condition (5b) is true. The possible weights with their meanings are shown in Table 1.

Table 1
Edge weights and their meaning

weight	relation between the variables		
0	we have not any information (not yet)		
1	critical condition (5a)		
2	not critical condition (5a)		
3			critical condition (5b)
4	critical condition (5a)	and	critical condition (5b)
5	not critical condition (5a)	and	critical condition (5b)
6			not critical condition (5b)
7	critical condition (5a)	and	not critical condition (5b)
8	not critical condition (5a)	and	not critical condition (5b)

Definition 3. (Associated graph of conditions) Let a basic or a modified or an extended BP programming problem be given. Let the number of the vertices of graph G be *n*, the same as the number of variables in the BP problem. Assign the variables of the BP problem to the vertices of the graph G. If variable A is on the left side of a condition of type (1) or type (2), then we will use arrows from A to the nodes representing variables on right hand side of that condition. The weights of these arrows depend on the condition. First we draw all edges that are needed. Then we will calculate their weights in the following way. Let all weights be 0 initially (we can draw all possible arrows in the graph with weight 0). After this, we read each condition one by one, and modify the weights according to the next steps:

- If variable A is on the LHS of a type (2) condition and variable B appears as a member of the product on the RHS and the weight of the corresponding arrow is not 1 mod 3, then the weight is increased or decreased by 1 such that it will be 1 mod 3.

- If variable A is on the LHS of a type (2) condition and variable B appears in the product written as (1 – B) on the RHS and the weight of the corresponding arrow is not between 3 and 5 (inclusively), then the weight is increased or decreased by 3 such that it becomes between 3 and 5.
- If variable A is on the LHS of a type (1) condition and variable B appears as a member of the product on the RHS and the weight of the corresponding arrow is divisible by 3 then its value is increased by 2.
- If variable A is on the LHS of a type (1) condition and variable B appears in the product on the RHS written in the form (1 – B) and the weight of the corresponding arrow is less than 3 then 6 is added to its value.

And now, using the type (3) conditions, we can assign values to some nodes from the set {0,1}. If variable A appears on the LHS of a condition of type (3a), then we assign 0 to it. If it occurs on the LHS of a condition of type (3b), then we assign the value 1 to it. If a variable is a member of the product of a type (3) condition (either 3a or 3b) on the right hand side, then we assign 1 to it, if A appears as (1–A) in the product on the RHS of any condition of type (3), then we assign 0 to it.
Then G is the (initial) graph of the BP problem.

We say that an assignment of the values {0,1} to the variables is a solution of the graph if it is compatible with all the conditions represented, i.e., all conditions are satisfied. ■

By Lemmas 1 and 2, Definition 2 and Table 1, one can see that all information about the conditions of the problem is encoded in the graph, and thus, the solution(s) of the graph and the BP problem coincide.

3 Manipulation of the Graph

In this section we give and explain the graph modifying steps. They are defined in such a way that the possible solutions do not change, hence, first, we define the equivalence among graphs.

Definition 4. (Graph-Equivalence) We say that two graphs are equivalent if the sets of the possible solutions of them (i.e. the possible solutions of the BP problems which are represented by these graphs) are the same. ■

In this part some possible steps for modifying the graph are shown. We will use some kinds of changing steps as node-evaluating and arrow-adding or changing.

There are two types of node-evaluating steps, let us see them first. We start with those that are used in some cases with an already known value of a node. They are called valuable arrows and listed in the following list.

Let A and B be two distinct vertices.

a) If A has value 1 and the arrow from A to B has a weight that is not a multiplier of 3, then let the value of B be 1.

b) If A has value 1 and the arrow from A to B has a weight that is larger than 2, then assign 0 to B.

c) If A has value 1 and the arrow starting at B and ending at A has a weight that is not less than 3, then assign 0 to B.

d) If A has value 0 and each arrow starting at A has a weight different from 4 and there is exactly one edge starting at A with odd weight, and its weight is either 1 or 7, and it goes to B, then let the value of B be 0.

e) If B has value 1 and each edge from A has a weight different from 4 and there is a unique edge starting at A with odd weight, and its weight is either 1 or 7 and it goes to B, then let 1 be assigned to A.

f) If B has value 0 and the arrow starting at A and ending at B has a weight that is not a multiplier of 3, then let the value of A be 0.

g) If B has value 0 and each arrow from A has a weight different from 4 and there is a unique arrow from A with odd weight, and its weight is either 3 or 5 and it goes to B, then assign 1 to A.

h) If B has value 0 and each arrow from B has a weight different from 4 and there is a unique edge starting at B with odd weight, and its weight is either 3 or 5 and this arrow ends at A, then let A have a value of 1.

The other types of node-evaluating steps are the so-called basic schemes. They work without any known values.

α) If a loop arrow occurs at vertex A such that its weight is larger than 2, then let 0 be assigned to A.

Actually, by viewing its structure, the following scheme is between the valuable arrows and the basic schemes.

β) If vertex A has a starting arrow with weight 4, 5, 7 or 8, then let A have the value 0.

γ) Let A, B and C be three vertices. If there are edges both from A and from B to C with weights non-divisible by 3 and there is an arrow from A to B with weight 3 or 5 such that there is no other arrow from A with weight 4 or with an odd weight, then let 1 be assigned to C.

We continue with arrow adding and arrow changing steps.

Considering two (A, B) or three vertices (A, B and C) in the graph such that they have the connections satisfying any of the following conditions, we can modify the weight of an arrow as it is specified below:

1) If the weight of the edge from A to B has a value more than 2 then
if the weight of the edge from B to A is at most 2, then it is increased by 6.

2) If each edge from A has a weight different from 4 and there is a unique edge from A with odd weight, moreover it is either 1 or 7, and it ends at B, then
if the weight of the edge from B to A is divisible by 3, then let it be 2 more than it was.

3) If none of the values of the weights of the edges A to B and B to C are divisible by 3, then,
if the weight of A to C is a multiplier of 3, then let this weight be increased by 2.

4) If the weight of the edge from A to B is not a multiplier of 3 and the weight of the edge from B to C is greater than 2, then

   - If the weight of the edge from A to C is at most 2, then it is increased by 6
   - If the weight of the edge from C to A is at most 2, then it is increased by 6

5) If the weight of the arrow from A to B is not a multiplier of 3 and the weight of the edge from C to B is not less than 3, then

   - If the weight of the edge from A to C is at most 2, then it is increased by 6
   - If the weight of the arrow from C to A is at most 2, then it is increased by 6

6) If the weight of the arrow from A to B is not less than 3 and each arrow from B has weight different from 4 and there is a unique edge from B with an odd weight, and it is either 1 or 7, and this edge ends at C, then

   - If the weight of the arrow from A to C is less than 3, then it is increased by 6
   - If the weight of the edge from C to A is less than 3, then it is increased by 6

7) If each edge from B has a weight different from 4 and there is a unique edge from B with odd weight and this weight is either 1 or 7, and this edge ends at C, and the weight of the arrow from B to A is not less than 3, then

   - if the weight of the arrow from A to C is less than 3, then it is increased by 6
   - if the weight of the edge from C to A is less than 3, then it is increased by 6

8) If the weight of the edge A to B is at least 3 and each edge from B has a weight different from 4 and there is a unique edge from B with odd weight and this weight is either 3 or 5, and this edge ends at C, then
if the weight of the arrow from A to C is a multiplier of 3 then it is increased by 2.

9) If the weight of the arrow from A to B is not less than 3 and each edge from C has a weight different from 4 and there is a unique edge from C with odd weight and this weight is either 3 or 5 and this edge ends at B, then
if the weight of the edge from A to C is a multiplier of 3, then it is increased by 2.

10) If each edge from B has a weight different from 4 and there is a unique edge from B with odd weight and this weight is either 3 or 5 and this edge ends at C, and the weight of the arrow from B to A is not less than 6, then
if the weight of the arrow from A to C is a multiplier of 3 then it is increased by 2.

11) If the weight of the arrow from B to A is not less than 3 and each edge from C has a weight different from 4 and there is a unique edge from C with odd weight and this weight is either 3 or 5, and this edge ends at B, then
if the weight of the edge from A to C is a multiplier of 3, then it is increased by 2.

We are continuing with 3 arrow-changing steps, however, these steps use already known values at some vertices.

12) If 1 is assigned to B and the weight of the edge from A to B is 1 mod 3 and there is another edge starting at A with either an odd weight or with weight 4, then
the weight of the edge from A to B is incremented by 1.

13) If 0 is assigned to B and the weight of the edge from A to B is either 3 or 5 and there is another edge from A either with weight 4 or with an odd weight, then
the weight of the edge from A to B is increased by 3.

14) If A has the value 1 and

- If the weight of an edge from A is 1, then let the weight of this edge be 2
- If the weight of an edge from A is 3, then let the weight of this edge be 6

Finally, we have some arrow-changing steps for subgraphs containing three vertices. Let A, B and C be three vertices.

15) If the value of the edge from A to B and the value of the edge from A to C are both 1 mod 3 and the weight of the edge from B to C is not a multiplier of 3, then
let the weight of the edge from A to C be increased by 1.

16) If the value of the edge from A to B and the value of the edge from A to C are both between 3 and 5 (inclusively) and the weight of the edge from B to C is not a multiplier of 3, then ,
the weight of the edge from A to B is increased by 3.

Now let us discuss these steps briefly. The next lemma shows how we can use the critical edges to gain information.

Lemma 3. If there is a unique critical condition for a vertex (exactly one of the weights of the edges from there is odd, i.e., has weight 1, 3, 5 or 7), then this condition must represent an equality.

Proof. It goes by indirect method. ■

For example, we use the previous lemma at arrow changing steps 6) or at node evaluating step d). Using this property and the meaning of atomic conditions (type (5a) and (5b) inequalities) represented by the edges of the graph, the justification of valuable arrow steps a) to h) are based on inferences to find the unique value for a variable represented at a node where the value of one of the involved variables are already known.

It is easy to check that at the basic scheme steps α), β) and γ), there is only one possibility for the specified variable to satisfy the conditions and it is assigned for the corresponding variable.

Condition (5a) is transitive and (5b) is symmetric. We use these properties in some steps, for example in 1), 2) etc. In those arrow-adding cases we increase the weight by 2 or 6 according to the new condition (5a) or (5b), and these new conditions are not critical, because they are not from an original type (2) condition. If a condition cannot be critical (see its definition) or our graph is equivalent to the original in the case of change a critical condition to non-critical one, then we use those steps which modify the given condition to non-critical one: we increase the respective value by 1 in case of (5a) or by 3 in case of (5b). Hence, steps 1) thru 16) are correct.

Remark 1. The arrows which are critical in the solution are also critical in the original graph.

4 Algorithm to Solve BP Problems

In this part, we present an algorithm which works in two variations. The first variation provides a relatively good solution in a short time by a greedy approach, if there is a possible solution of the problem. The second variation works more slowly: it includes the case when we do not stop after the first solution is found. This variation of the algorithm will find the optimal solution, because it works after the first solution is found till it is guaranteed that no better solution can be found than the latest one that has already been found.

Without loss of generality, for simplicity, we may assume that the variables indexed in non-decreasing order by the absolute values of their coefficients (multipliers) in the goal function, i.e. $| a_i | \geq | a_j |$ if and only if $i < j$ for all i,j between 1 and n.

Algorithm 1 searches in almost the whole tree of the state space. It is a mixture of the known knapsack and backtrack [4, 9, 19, 21] algorithms extended with our graph-modifying method.

Algorithm 1

Input: a BP problem.
Output: a solution/the best solution (or "Contradiction" if no solution exists).
0. If the goal is to minimize Z then let Z' = – Z be the new goal function, and maximize it. (Else let Z' = Z.)
1. Draw the graph representation of the BP problem.
2. Apply the possible graph-changing steps.
3. If there is a vertex with both values (1 and 0), then the problem is not solvable. (Contradiction.) STOP
4. If there is a unique value at each vertex, then this assignment could be the solution. Check it (because using graph-changing steps we may get a contradiction; see previous step)). If it satisfies all the conditions, it is the solution; otherwise the problem is not solvable (Contradiction). STOP
5. If there are no more usable graph-changing steps, then choose the variable which has the smallest index among the variables which do not have any assigned value yet. If the multiplier is positive than choose and assign value 1 to this vertex, if the multiplier is negative, then assign value 0 for this variable.
6. Use the possible graph-changing steps.
7. If there is a vertex with both values (1 and 0), then this is contradiction. (BACKTRACK)
If there is a unique value at each vertex, then this may be a solution. (Check it! and if it is fairly good, then STOP, else memorize this solution (if this is better than the previously found solution) and BACKTRACK.)
8. If BACKTRACK then go back to the previous value assigning in step 5, and find the last chosen variable for which we have not tried both values, and assign the other value to it. Cancel all the graph-changing steps done after the step we have assigned the previous value to this variable. If such a variable does not exist, then we have already tried all possibilities and finished the search, if there is a memorized solution, then the last one is the best, else there is no possible solution. STOP
9. Go to step 5.

We can use cuts to speed up the algorithm. If we already have a solution, then we can check the possible maximal value of Z' for the remaining part of the state space:

Let our goal be, to maximize the function $Z' = a_1 \cdot B_1 + a_2 \cdot B_2 + ... + a_n \cdot B_n$

We call the values B_i fixed values if they cannot change in the remaining part of the search. We use the concept critical node of the search tree, for which we gave the value of a variable by step 5 of the algorithm, and it is the variable with the smallest index such that we have not tried with both values. The fixed values are the variables which have got values before we used the value assigning at the critical node. (The fixed values are the variables which have smaller indices than the variable at the critical node and the variables which got their values by using graph-changing steps using only nodes with other fixed values.)

The possible best solution in the remaining state tree is the following: if the fixed values have their actual values and all other variables has the best value (i.e. 1, if the multiplier is positive and 0 if the multiplier is negative.) Therefore, if we remember the fixed values we can easily calculate that value:

$$\sum_{\substack{\text{for } i, \text{ where} \\ B_i \text{ fixed}}} a_i B_i + \sum_{\substack{\text{for i, where} \\ B_i \text{ not fixed,} \\ \text{and } a_i > 0}} a_i$$

If this amount is not greater than the best found (memorized) solution, then we can finish the search. We can calculate this amount after we make a BACKTRACK. Or we can evaluate this sum after all node-value assigning (at step 5 or via graph-changing steps) and we can use BACKTRACK earlier if this value is already less than at the best found solution. If a variable with negative multiplier get the value 1 or a variable with positive multiplier get the value 0, then the sum will decrease by the absolute value of the multiplier.

As a consequence of the description given in Section 3, the graph-changing part of our algorithm works properly. Step 5 of the algorithm represents the knapsack problem and we use greedy algorithm to solve it. Since this step cannot guarantee the (best) solution we expand our algorithm by the backtrack method. We use a kind of branch-cost backtrack, in which we can calculate the maximal value of the goal function of the possible solutions of the remaining search space and we can use this value to decide whether we continue the search.

Remark 2 (On the difference between various types of BP) By analyzing the original graphs of various types of BP-problems one can see that in basic problems only even numbers are used as weights in the graph (2, 6 and 8). In modified BP-problems we use more weights, and in extended problems we have knowledge about values of some nodes originally.

5 Examples

In this part we will show some examples.

Example 1 (BP problem without possible solution)

Let A, B and C be the variables. The conditions:
$A < B$
$1 - B < C$
$B \leq A \cdot C$
$Z = 5A + 3B + C$ and the task is to minimize Z.

Solution. It is an extended BP problem. We want to maximize the function $Z' = -5A - 3B - C$. First we draw the graph of the example (Fig. 1). The first condition gives the values of A and B (0 and 1, respectively). The second

inequality gives the value B and C (both of them are 1). And we have two weighted edges from the third condition.

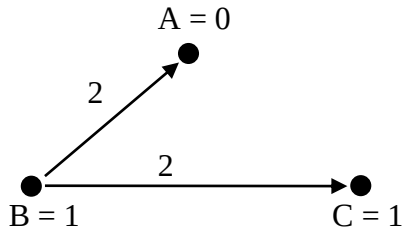

Figure 1
Graph representation of Example 1

We have a value at each node, but it is contradictory. (Between B and A, the weight of the arrow means that it is impossible that B is 1 and A is 0, it is described in point f) with interchanged role of A and B.)

Example 2

$A \leq D$
$A = 1 - B$
$B = D \cdot (1 - E)$
$C \leq (1 - B) \cdot D$
$E \leq B \cdot F$
$Z = 3A - 2.21B + 3D - 22E - 3.25F$, the task is to maximize value of Z.

Solution. This is a modified BP problem. Let us draw its graph (Fig. 2).

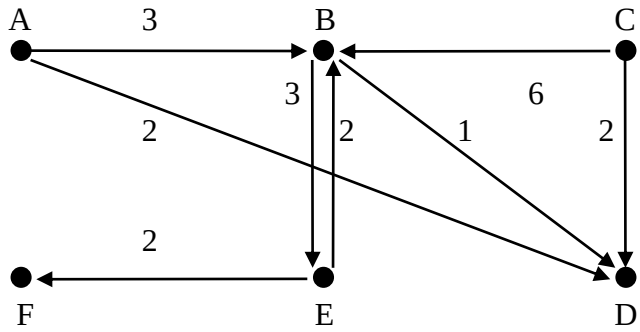

Figure 2
Graph of Example 2

In this graph, one can use some local graph changing steps: There is a basic scheme γ) with vertices A, B and D, consequently, D = 1. Then one can use step 12) for the edge between B and D, thus its weight becomes 2. One can use step 1)

between B and E, therefore the weight of the edge EB goes to 8. But according to step β) value 0 is obtained at E. Now one can use step g) for the arrow BE getting value 1 at vertex B. Then, step c) for A and B and for B and C are used to get values at A and C: they both receive value 0. The values of five vertices are already known. There is no step to get the value of F. We are at point 5 of the algorithm. The goal is to maximize Z, in which F has negative coefficient. Thus, let F = 0. This is the best possible solution: B = D = 1 and A = C = E = F = 0, yielding Z = 0.79.

Example 3

$A \leq (1 - E)$
$B = (1 - C) \cdot (1 - D)$
$C \leq A \cdot D$
$E < A \cdot (1 - E)$
$E = B \cdot C \cdot (1 - D)$
$Z = -5A + 9B + 3C + 7D + 0.5E$

The goal is to maximize Z.

Solution. We will use the next ordering on the variables: B, D, A, C and E, and function Z' = Z. It is an extended BP problem. The graph of these conditions is shown in Fig. 3.

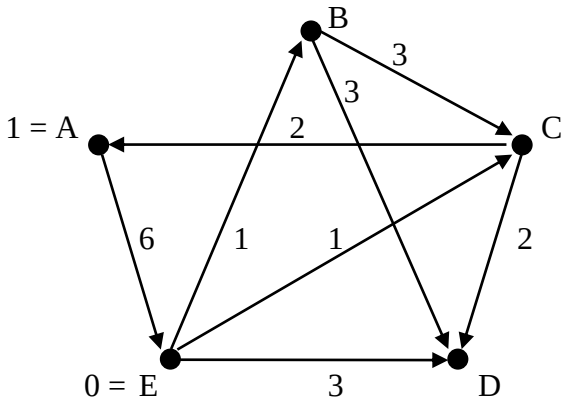

Figure 3
The graph of conditions in Example 3

From the fourth condition we know the values of two variables. There are some arrow-changing steps 3) and 4) for arrows starting from E and from B to C we can use step 16) etc., but we have no information about the values of the nodes B, C and D. See Fig. 4.

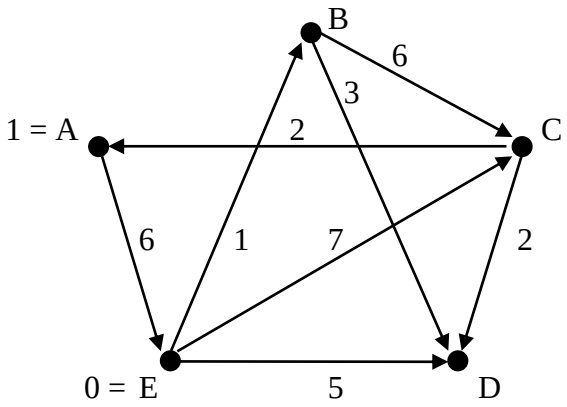

Figure 4
The graph of example 3 after changing arrows

Now we are at point 5 of the algorithm. A and E are fixed values. Let us choose 1 for the value of B. We can use step a) and we get C = 0 and D = 0. One can check it; it is a solution with Z = 4. Now we want the best solution, thus we make a BACKTRACK and let B = 0. Now we check the value Z of the remaining possible best solution: A and E were fixed before B got the value and we are trying the second value for B, so they are fixed. (It is easy to check that there are no more fixed values.) Therefore, the amount is -5 + (3+7) = 5, which is greater than the value of our memorized solution, thus we continue the search: We have no graph-changing steps to determine the values of C and D, so we are at point 5 of the algorithm and let D = 1. We cannot get the value of C by graph-changing, so we are at point 5 again, let C = 1. We can check, it is also a solution, with the value Z = 5. We memorize it, and we finish the search because we know from the previous calculation that it is impossible to get a better solution.

Conclusions

In this paper, various types of Boolean Programming problems have been considered. Using graph-theoretical approach we have solved these special 0-1 integer programming problems. In practice, we can approach similar problems, if we have conditions, for switches or Boolean circuits. We can solve such problems in which each variable is binary, i.e., it has a value of a characteristic function. In some logical exercises and puzzles the conditions are similar to the conditions investigated here [13, 15-17], or we can write them in the form of the conditions of BP problems and we can use graphs to solve them [11, 14]. We used linear goal-function in the optimization. The conditions are strong (i.e. strict) and weak inequalities and equations. In the graph representation, the critical edges, are used to represent the non-linear conditions. Our algorithm is based on local information: we can modify the graph by changing the weight values of the edges and by assigning values to nodes. Interesting properties of graphs are noted, such

as the arrows with weights non-divisible by 3 are specifying a transitive relation among nodes, while the arrows with weights at least 3 are representing a symmetric relation. In the future, we expect some more interesting phenomena by a more detailed analysis of our theory. Our method, similarly to knapsack algorithms, can give a relatively good solution in a short time in many cases. The algorithm uses backtracking graph-search strategy to find also the optimal solution of these problems.

We can use our algorithm for the case of arbitrary goal-functions, in step 5, we need to choose the most 'important' variable to be 1 (or the variable which is the least 'important' to be 0), where the importance property is specified based on the goal function. Our method can easily be implemented by using matrices of the graphs.

When we allow more than one type (2) condition for a variable in the LHS, we must use and-or graphs (i.e., hypergraphs) to represent the conditions. We hope that a variation of the presented method will work for other integer-valued programming problems for which we allow more values than 2 for the variables.

Finally, we note that the technique used here is related to methods to solve logical puzzles [14]. On the other hand, SAT solvers [1, 6, 8] provide valuations of the variables such that the given formula evaluates to true, if it is possible. The SAT problem is one of the most known NP-complete problems. SAT solvers can also be used to solve certain puzzles. Moreover, the problem of finding a satisfying valuation of a logical formula that optimizes a linear function of variables is called MinCostSAT [5]. Thus, we can see that Boolean Programming is an important and challenging topic with various new approaches.

Acknowledgements. The author is very thankful for the anonymous reviewers for their constructive comments helping to improve the paper.

References

[1] Bengt Aspvall, Michael F. Plass, Robert Endre Tarjan: A linear-time algorithm for testing the truth of certain quantified Boolean formulas, Information Processing Letters 8 (1979) 121-123

[2] Béla Bollobás: Graph Theory: An Introductory Course. Springer-Verlag, New York, 1979

[3] Peter Barth: Linear 0-1 Inequalities and Extended Clauses, Operations Research '93 (Springer) pp. 24-27

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: Introduction to Algorithms. MIT Press and McGraw-Hill, 1990 (3rd edition, 2009)

[5] Zhaohui Fu, Sharad Malik: Solving the minimum-cost satisfiability problem using SAT based branch-and-bound search. In Proceedings of the

2006 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '06) 2006, San Jose, CA, 2006, pp. 852-859

[6] Weiwei Gong, Xu Zhou: A survey of SAT solver. AIP Conference Proceedings 1836, 020059 (2017) https://doi.org/10.1063/1.4981999

[7] Peter L. Ivanescu, Sergiu Rudeanu: Boolean methods in operations research and related areas. Econometrics and Operations Research, Vol. VII, Springer-Verlag New York, Inc., New York 1968

[8] Gábor Kusper, Csaba Biró: Solving SAT by an Iterative Version of the Inclusion-Exclusion Principle. 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2015) Timisoara, 2015, pp. 189-190

[9] Ken McAloon, Carol Tretkoff: Optimization and Computational Logic, Wiley-Interscience series in discrete mathematics and optimization, 1996

[10] Benedek Nagy, Márk Kósa: Logical puzzles (Truth-tellers and liars), ICAI'01, Eger, 2001, pp. 105-112

[11] Benedek Nagy: Boole programozás gráfok segítségével (in Hungarian, Boolean programming and related graphs) Szigma 23 (2002) 115-130

[12] Benedek Nagy: Truth-teller-liar puzzles and their graphs, Central European Journal of Operations Research 11 (2003) 57-72

[13] Benedek Nagy: SW-type puzzles and their graphs, Acta Cybernetica 16 (2003) 67-82

[14] Benedek Nagy: Boolean programming, truth-teller-liar puzzles and related graphs, ITI 2003: 25th International Conference on Information Technology Interfaces, Cavtat, Croatia (2003) 663-668

[15] Benedek Nagy: Duality of logical puzzles of type SW and WS - their solution using graphs, Pure Mathematics and Applications 15 (2005) 235-252

[16] Benedek Nagy: SS-típusú igazmondó-hazug fejtörők gráfelméleti megközelítésben (in Hungarian, SS-type truthteller-liar puzzles and their graphs), Alkalmazott Matematikai Lapok 23 (2006) 59-72

[17] Benedek Nagy, Gerard Allwein: Diagrams and Non-monotonicity in Puzzles, Diagrams'2004, LNCS, LNAI 2980 (2004) Cambridge, England, 82-96

[18] András Prékopa: Studies on mathematical programming, Mathematical Methods of Operations Research, Vol. 1, Budapest, Akadémiai Kiadó 1980

[19] Steven S. Skiena: The Algorithm Design Manual, Springer-Verlag, New York, 1997

[20] Raymond Smullyan: Forever Undecided, Alfred A. Knopf, New York, 1987

[21] Stuart Russell, Peter Norvig: Artificial Intelligence – A Modern Approach. Prentice Hall, 1995 (3rd edition, 2009)

[22] Béla Vizvári: On the optimality of the greedy solutions of the general knapsack problems, Optimization 23/2 (1992) 125-138