Restricted OPB Format in Use in the PB Competitions

Olivier ROUSSEL

roussel@cril.fr

Version of this document

The version number and the date of this document (as recorded by the versioning system) are given below. They let you identify quickly if you have the most recent version of this document.

```
Version: $Rev: 4524 $
Last modification: $Date: 2024-03-11 18:52:57 +0100 (Mon, 11 Mar 2024) $
```

This document is based on the description of the OPB format used in the PB16 competition [RM16], with the following main changes :

- separation of the description of the general OPB format from the description of the restricted format used in the competition
- introduction of the *#equal=* hint to facilitate constraints allocation
- introduction of the *intsize=* hint to let the solver decide if it can deal with the integers used in the formula

1 Summary of the simplifications applied to the general OPB format

In order to facilitate the participation of a solver to the competition, the general OPB format is not used, but instead, a simplified, restricted version. This simplified format is a subset of the general format. High quality provers are encouraged to parse the general format, which is more user-friendly.

The main simplifications and restrictions are presented below:

- Only ASCII characters may be used in the file. Other Unicode characters are forbidden.
- The objective function can only be *min:*.

- A Boolean variable (atom) always starts by a lowercase 'x' immediately followed by a strictly positive integer number. The integer number can be considered as an identifier of the variable. This integer identifier is strictly less than 2^{32}. Therefore, a solver can input a variable name by reading a character (to skip the 'x') and then a 32 bits integer.

- Variable names are guaranteed to range from "x1" to "xN" where N is the total number of variables in the instance (as given on the first line of the file). Each variable between x1 and xN will occur in at least one constraint or the objective function.

- Each variable present in the objective function will occur in at least one constraint.

- The negation of an atom A ($\sim A$) will not appear in a linear pseudo-Boolean file (it will be translated to 1-A).

- The only relational operators used are $>=$ and $=$. These are always written with ASCII characters.

- As a hint to perform memory allocation, the first line of a linear instance will be a comment containing a sequence of keyword-value pairs. A space separates the keyword and the value to make parsing trivial. The separator between pairs is a space. The list of keywords, in order, is the following (new keywords are shown in bold face):

  1. "#variable=" is followed by a space and the number of variables in the file,
  2. "#constraint=" followed by a space and the number of constraints in the file,
  3. **"#equal=" followed by a space and the number of "strictly equal" constraints in the file,**
  4. **"intsize=" followed by a space and the number of bits required to represent, for any constraint of the formula, the sum of the absolute value of all integers that appear in the constraint.** Formally, let C_i be a constraint $\sum_j c_j^i.x_j \geq d$ (where $\geq$ may be replaced by $=$). If there is an objective function, it is seen as a pseudo constraint with a null degree ($d = 0$). The value $size$ that appears after "intsize=" is given by $size = max_i(1 + \lfloor log_2(|d| + \sum_j |c_j^i|) \rfloor)$. A solver which uses integers of at least $size$ bits (plus one for signed integers) and that simply checks the satisfaction of constraints by summing the coefficient of all true literals and subtracting the degree of the constraint will never face integer overflow. Solvers that perform more clever computations (such as PB constraints learning) should be more careful.

- For a non-linear instance, the first line will be a comment containing the sequence of keyword-value pairs defined for linear instances, followed by the pairs given below, in order:

1. "#product=" is followed by a space and the number of different products of variables present in the file
2. "sizeproduct=" is followed by a space and the total number of literals which appear in different products.

These two informations allow the parsers to compute the total number of linear constraints that will be passed to the solver when the parser is asked to linearize the formula. The keyword "#product=" also clearly indicates that this is a non-linear instance.

- Variables that appear inside a product are guaranteed to be ordered from the lowest to the greatest index.
- In the WBO tracks of PB competition, the top cost will always be specified. When all interpretations are admissible (i.e. when $T = \infty$), T will be given as 1 plus the sum of each soft constraint cost.

2 Introduction

This document presents the restricted OPB format, which is intended to be used only in the PB competitions. A high quality pseudo-Boolean solver is expected to support the general OPB format, which is slightly more complex, but more user-friendly. In constrast, the restricted format is solver-friendly, that is, it simplifies the implementation of a PB solver.

The initial OPB format was described at the end of the README file in the distribution of OPBDP `http://opbdp.sourceforge.net/`. It was subsequently extended and modified for the needs of the PB competitions (from PB05 `https://www.cril.univ-artois.fr/PB05/` to PB16 `https://www.cril.univ-artois.fr/PB16/`).

The current version of the OPB format allows to express two different kinds of constraints (linear and non-linear) and three different problems: PBS (Pseudo-Boolean Satisfaction), PBO (Pseudo-Boolean Optimization), WBO (Weighted-Boolean Optimization). The OPB format allows to express constraints with integer coefficients of arbitrary size.

3 Input Format

This section details the input format for

- Pseudo-Boolean Satisfaction (PBS) instances (Section 4.1)
  the solver must find an interpretation which satisfies each constraint (decision problem)
- Pseudo-Boolean Optimization (PBO) instances (Section 4.1)
  the solver must find an interpretation which satisfies each constraint and minimizes the value of an objective function (optimization problem)

- Weighted Boolean Optimization (WBO) instances (Section 4.2)
  the solver must find an interpretation which minimizes the cost of violated constraints (optimization problem)

4 Size of integers

A pseudo-Boolean formula may use integers of arbitrary size, because this can simplify the encoding of different problems (such as the factorization problem). A solver that uses fixed precision integers (such as 32 or 64 bits integers) may be faced with two problems:

- a coefficient in the formula may be larger than the maximum value of the integer type used by a solver. This is rather easy to check at parse time.
- when a solver combines different constraints (with addition for example), it may exceed the largest integer that can be represented (integer overflow). This is much harder to detect in general.

A simple solution is to use an arbitrary precision library such as GNU Multiple Precision Arithmetic Library for C/C++ (`http://gmplib.org/`) or the BigInteger class in Java (`https://docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html`). This has an obvious computational cost, but is the only way to support any possible instance. Another policy is to detect such integer overflows, or to avoid operations that could cause integer overflows.

In the past PB competitions, a SMALLINT category of instances was defined, and was supposed to limit the possibility of an integer overflow. However, there was no guarantee since this depends on the techniques used by the solver (e.g. learning). Besides, the SMALLINT category was defined by a limit of 2^{21} on the sum of coefficients, but this limit does not make sense for all solvers. Indeed, a single limit cannot fit all the solvers limitations.

Therefore, it is now the solver sole responsibility to check if it can handle the coefficient used in a PB formula. If it cannot, it must print 's UNSUPPORTED' at parse time to indicate that it cannot deal with the instance. The "intsize=" hint given on the first line of the instance can help the solver take a decision.

4.1 Pseudo-Boolean Satisfaction (PBS) and Pseudo-Boolean Optimization (PBO)

A PBO file and a PBS file only differ by the presence of an objective function (starting with the keyword `min:`) at the beginning of the file.

Lines beginning with a star are comments and can appear anywhere in the files. A PBS file contains a list of pseudo-Boolean constraints. Each pseudo-Boolean (PB) constraint consists of a left side (a list of weighted Boolean terms), a comparison operator and a right side which is an integer constant. A PBO file contains an objective function followed by the constraints.

In the PB competitions, an OPB file is written in pure ASCII. Since PB constraints can be normalized, only two comparison operators are allowed (greater than, equal) and PBO instances only use the *min:* keyword in the objective function.

The syntax of these files can be described by a simple BNF grammar (see `http://en.wikipedia.org/wiki/Backus-Naur_form`). `<formula>` is the start symbol of this grammar.

```
<formula>::=
    <sequence_of_comments>
    [<objective>]
    <sequence_of_comments_or_constraints>

<sequence_of_comments>::=
    <comment> [<sequence_of_comments>]

<comment>::=
    "*" <any_sequence_of_characters_other_than_EOL> <EOL>

<sequence_of_comments_or_constraints>::=
    <comment_or_constraint>
    [<sequence_of_comments_or_constraints>]

<comment_or_constraint>::=
    <comment>|<constraint>

<objective>::=
     <objective_type> <zeroOrMoreSpace> <sum>  ";"

<constraint>::=
    <sum> <relational_operator>
    <zeroOrMoreSpace> <integer> <zeroOrMoreSpace> ";"

<sum>::=
    <weightedterm> | <weightedterm> <sum>

<weightedterm>::=
    <integer> <oneOrMoreSpace> <term> <oneOrMoreSpace>

<integer>::=
    <unsigned_integer>
    | "+" <unsigned_integer>
    | "-" <unsigned_integer>

<unsigned_integer>::=
    <digit> | <digit><unsigned_integer>


<objective_type>::=
    "min:"

<relational_operator>::=
    "=" | ">="

<variableName>::=
    "x" <unsigned_integer>
```

```
<oneOrMoreSpace>::=
    " " [<oneOrMoreSpace>]

<zeroOrMoreSpace>::=
    [" " <zeroOrMoreSpace>]
```

The input format allows the specification of both linear and non-linear pseudo-Boolean instances. The definition of `<term>` will be given in the corresponding subsections.

This grammar let us write a very simple parser and avoid some ambiguities present in the original description of the OPB format. At the same time, the format remains easily human readable and is mostly compatible with solvers using the OPB format.

Integers may be of arbitrary size (i.e. contain any number of digits) (see section 4).

Some details:

- A line starting with a '*' is a comment and can be ignored. Comment lines are allowed anywhere in the file.
- Each non comment line must end with a semicolon ';'
- In a PBO instance, the first non comment line is an objective function to minimize. It starts with the word "min:" followed by the linear function to minimize and is terminated by a semicolon. No other objective function can be found after this first non comment line.
- A constraint is written on a single line and is terminated by a semicolon.
- A Boolean variable (atom) is named by a lowercase 'x' followed by a strictly positive integer number. The integer number can be considered as an identifier of the variable. This integer identifier is strictly less than 2^{32}. Therefore, a solver can input a variable name by reading a character (to skip the 'x') and then a 32 bits integer.
- Variable names are guaranteed to range from "x1" to "xN" where N is the total number of variables in the instance (as given on the first line of the file). Each variable between x1 and xN will occur in at least one constraint or the objective function.
- Each variable present in the objective function will occur in at least one constraint.
- Each variable name must be followed by a space
- The weight of a variable may contain an arbitrary number of digits. There must be no space between the sign of an integer and its digits.
- Lines may be very long. Programmers should avoid reading a line as a whole.
- PBS and PBO files are expected to have a ".opb" extension.

4.1.1 Linear instances

For linear pseudo-Boolean instances, `<term>` is defined as

```
<term>::=<variableName>
```

The negation of an atom A will not appear in a linear pseudo-Boolean file (it will be translated to 1-A).

As a hint to perform memory allocation, the first line of a linear instance will be a comment containing a sequence of keyword-value pairs. A space separates the keyword and the value to make parsing trivial. The separator between pairs is a space. The list of keywords, in order, is the following (new keywords are shown in bold face):

1. "#variable=" is followed by a space and the number of variables in the file,
2. "#constraint=" followed by a space and the number of constraints in the file,
3. **"#equal=" followed by a space and the number of "strictly equal" constraints in the file,**
4. **"intsize=" followed by a space and the number of bits required to represent, for any constraint of the formula, the sum of the absolute value of all integers that appear in the constraint.** Formally, let C_i be a constraint $\sum_j c_j^i.x_j \geq d$ (where $\geq$ may be replaced by $=$). If there is an objective function, it is seen as a pseudo constraint with a null degree ($d = 0$). The value $size$ that appears after "intsize=" is given by $size = max_i(1+\lfloor log_2(|d|+\sum_j |c_j^i|)\rfloor)$. A solver which uses integers of at least $size$ bits (plus one for signed integers) and that simply checks the satisfaction of constraints by summing the coefficient of all true literals and subtracting the degree of the constraint will never face integer overflow. Solvers that perform more clever computations (such as PB constraints learning) should be more careful.

Examples

```
* #variable= 5 #constraint= 4 #equal= 1 intsize= 61
*
* this is a dummy instance
*
min: 1 x2 -1 x3 ;
1 x1 +4 x2 -2 x5 >= 2;
-1  x1 +4   x2 -2   x5 >= +3;
1234567890123456789 x4 +4 x3 >= 10;
* an equality constraint
2 x2 +3 x4 +2 x1 +3 x5 = 5;
```

4.1.2 Non-Linear Instances

The format for non-linear pseudo-Boolean instances is a straightforward generalization of the linear format. The main changes are:

- Both in the objective function and in the constraints, the input format allows the specification of product of literals. Since literals are assigned values from $\{0, 1\}$, a product of literals is interpreted as 1 if and only if all of its literals are assigned to 1. In Boolean terms, a product represents a conjunction of literals.
- Products contain literals instead of variables. Therefore, variable names can be preceded with the character '$\sim$' in order to specify the negative literal of that variable

With these generalizations, it is possible to specify constraints like:

```
3 x1 x2 + 2 ~x3 ~x4 ~x5 -3 x6 >= +2 ;
```

If we would not allow negative literals in the format, we would have to replace this simple term

```
2 ~x3 ~x4 ~x5
```

with the significantly longer expression

```
-2 x3 -2 x4 -2 x5 +2 x3 x4 +2 x3 x5 +2 x4 x5 -2 x3 x4 x5 +2
```

For non-linear instances, <term> is defined as

```
<term>::=
    <oneOrMoreLiterals>

<oneOrMoreLiterals>::=
    <literal> | <literal> <oneOrMoreSpace> <oneOrMoreLiterals>

<literal>::=
    <variableName> | "~"<variableName>
```

There are further restriction enforced in the PB competitions:

- A product which contains one single literal will not use a negation (the negation $\sim$L will be replaced by 1-L during the normalization process)
- For a non-linear instance, the first line will be a comment containing the sequence of keyword-value pairs defined for linear instances, followed by the pairs given below, in order:
  1. "#product=" is followed by a space and the number of different products of variables present in the file
  2. "sizeproduct=" is followed by a space and the total number of literals which appear in different products.

  These two informations allow the parsers to compute the total number of linear constraints that will be passed to the solver when the parser is asked to linearize the formula. The keyword "#product=" also clearly indicates that this is a non-linear instance.
- Variables that appear inside a product are guaranteed to be ordered from the lowest to the greatest index. High quality provers are encouraged to avoid relying on this assumption as it may not hold outside the competition environment.

Examples The first example only illustrates the syntax (it does not encode any concrete problem).

```
* #variable= 5 #constraint= 4 #equal= 1 intsize= 64 #product= 5 sizeproduct= 13
*
* this is a dummy instance
*
min: 1 x2 x3 -1 x3 ;
1 x1 +4 x1 ~x2 -2 x5 >=2;
-1 x1 +4 x2 -2 x5 >= 3;
123456789012345678 90 x4 +4 x3 >= 10;
2 x2 x3 +3 x4 ~x5 +2 ~x1 x2 +3 ~x1 x2 x3 ~x4 ~x5 = 5 ;
```

This second example encodes a factorization problem (see the comments for the details). For lack of space, the last constraint has been printed on 2 lines but is actually stored on a single line in the file.

```
* #variable= 6 #constraint= 3 #equal= 1 intsize= 7 #product= 9 sizeproduct= 18
*
* Factorization problem: find the smallest P such that P*Q=N
* P is a 3 bits number (x3 x2 x1)
* Q is a 3 bits number (x6 x5 x4)
* N=35
*
* minimize the value of P
min: +1 x1 +2 x2 +4 x3 ;
* P>=2 (to avoid trivial factorization)
+1 x1 +2 x2 +4 x3  >= 2 ;
* Q>=2 (to avoid trivial factorization)
+1 x4 +2 x5 +4 x6  >= 2 ;
* P*Q=N
+1 x1 x4 +2 x1 x5 +4 x1 x6 +2 x2 x4 +4 x2 x5 +8 x2 x6 +4 x3 x4
                                              +8 x3 x5 +16 x3 x6 = 35;
```

This formula can easily be linearized and transformed into the following linear formula. The parsers provided for the competition are able to perform this linearization automatically.

```
* #variable= 15 #constraint= 21 #equal= 1 intsize=7
*
* linearized version of the factorization problem (P*Q=35)
* this linearization can be automatically done by the parsers we
* provide
min: +1 x1 +2 x2 +4 x3;
+1 x1 +2 x2 +4 x3 >= 2;
+1 x4 +2 x5 +4 x6 >= 2;
+1 x7 +2 x8 +4 x9 +2 x10 +4 x11 +8 x12 +4 x13 +8 x14 +16 x15 = 35;
* new variables introduced to represent the products
+1 x7 -1 x1 -1 x4 >= -1;
-2 x7 +1 x1 +1 x4 >= 0;
+1 x8 -1 x1 -1 x5 >= -1;
-2 x8 +1 x1 +1 x5 >= 0;
+1 x9 -1 x1 -1 x6 >= -1;
-2 x9 +1 x1 +1 x6 >= 0;
+1 x10 -1 x2 -1 x4 >= -1;
-2 x10 +1 x2 +1 x4 >= 0;
```

```
+1 x11 -1 x2 -1 x5 >= -1;
-2 x11 +1 x2 +1 x5 >= 0;
+1 x12 -1 x2 -1 x6 >= -1;
-2 x12 +1 x2 +1 x6 >= 0;
+1 x13 -1 x3 -1 x4 >= -1;
-2 x13 +1 x3 +1 x4 >= 0;
+1 x14 -1 x3 -1 x5 >= -1;
-2 x14 +1 x3 +1 x5 >= 0;
+1 x15 -1 x3 -1 x6 >= -1;
-2 x15 +1 x3 +1 x6 >= 0;
```

4.2 Weighted Boolean Optimization (WBO)

The Weighted Boolean Optimization (WBO) framework extends the MaxSAT concepts to pseudo-Boolean constraints. In [MMSP09], the WBO framework is defined as follows:

> A Weighted Boolean Optimization (WBO) formula ϕ is composed of two sets of pseudo-Boolean constraints, ϕ_s and ϕ_h, where ϕ_s contains the soft constraints and ϕ_h contains the hard constraints. For each soft constraint $\omega_i \in \phi_s$ there is an associated integer weight $c_i > 0$. The WBO problem consists in finding an assignment to the problem variables such that all hard constraints are satisfied and the total weight of the unsatisfied soft constraints is minimized (i.e. the total weight of satisfied soft constraints is maximized).

The WCSP (Weighted Constraint Satisfaction Problem) uses the notion of top cost T. All interpretation with a cost greater than or equal to T are non admissible and cannot be a solution. This notion is quite natural in practice. Therefore, we choose to refine the definition of the WBO formalism as follows:

> The WBO problem consists in finding an assignment to the problem variables such that all hard constraints are satisfied and the total weight of the unsatisfied soft constraints is minimized *and strictly less than* T.

A WBO instance does not contain an objective function. The first non comment line of the file starts with the keyword "soft:", followed by the top cost T and a semi-colon. If $T = \infty$, the top cost can be omitted and the first line is just "soft: ;"

A WBO file is expected to have a '.wbo' extension.

In the PB competition, the top cost will always be specified. When all interpretations are admissible (i.e. when $T = \infty$), T will be given as 1 plus the sum of each soft constraint cost.

In the rest of the file, hard constraints use the same syntax as in PBS files. Soft constraints are prefixed with their cost written in square brackets. As an example, "+1 x1 +2 x2 >= 2 ;" is a hard constraint and "[5] +1 x1 +2 x2 >= 2 ;" is a soft constraint which has a cost of 5.

The differences between the grammar of a PBS/PBO file and the grammar of a WBO file are small and detailed below.

```
<softformula>::=
    <sequence_of_comments>
    <softheader>
    <sequence_of_comments_or_constraints>

<softheader>::=
    "soft:" [<unsigned_integer>] ";"

<comment_or_constraint>::=
    <comment>|<constraint>|<softconstraint>

<softconstraint>::=
    "[" <zeroOrMoreSpace> <unsigned_integer> <zeroOrMoreSpace> "]"
    <constraint>
```

The cost of a soft constraint can be any positive integer (including big integers) as long as it is strictly less than T (otherwise the constraint would be hard).

A WBO instance can be transformed into a PBO instance by introducing extra variables which allow to neutralize soft constraints and adding an objective function which requires to minimize the cost of neutralized soft constraints.

In the PB competition, it is guaranteed that the first line of the file will be a comment that will start with the usual information given for a linear, or non-linear file. This line will then contain the following key-value pairs (separated by a space):

- `#soft=` followed by a space and the number of soft constraints in the file,
- `mincost=` followed by a space and the smallest cost of a soft constraint,
- `maxcost=` followed by a space and the greatest cost used in the file,
- `sumcost=` followed by a space and the sum of the soft constraints costs.

The value of `sumcost=` is used in place of the sum of the objective function coefficients to compute the value of `intsize=`.

As an example, a WBO instance with only linear constraints could start with the following line:

```
* #variable= 15 #constraint= 21 #equal 0 intsize= 20 #soft= 5 mincost= 1 maxcost= 1
  sumcost= 5
```

A WBO instance with non-linear constraints could start with the following line:

```
* #variable= 15 #constraint= 21  #equal 0 intsize= 20 #product= 5 sizeproduct= 13
  #soft= 5 mincost= 1 maxcost= 2 sumcost= 9
```

In the PB competition, it is also guaranteed that the costs used in the file will be as small as possible (but still may be of arbitrary size if this is needed to encode the problem). This implies that at least one cost will be 1, and if all costs are equal, their value will be 1.

4.2.1 Examples

Example 1 The optimal solution of the instance below is $x_1 = 0$ and has a cost of 2.

```
soft: 6 ;
[2] +1 x1 >= 1 ;
[3] -1 x1 >= 0 ;
```

Example 2 The optimal solution of the instance below is $x_1 = 0, x_2 = 1$ and has a cost of 2.

```
soft: 6 ;
[2] +1 x1 >= 1 ;
[3] +1 x2 >= 1 ;
-1 x1 -1 x2 >= -1 ;
```

Example 3 The instance below has no solution at all (the minimal cost is 6 which is not admissible).

```
soft: 6 ;
[2] +1 x1 >= 1 ;
[3] +1 x2 >= 1 ;
[4] +1 x3 >= 1 ;
[5] +1 x4 >= 1 ;
-1 x1 -1 x2 >= -1 ;
-1 x3 -1 x4 >= -1 ;
```

References

[MMSP09] Vasco Manquinho, Joao Marques-Silva, and Jordi Planes, *Algorithms for Weighted Boolean Optimization*, SAT '09: Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (Berlin, Heidelberg), Springer-Verlag, 2009, pp. 495–508.

[RM16] Olivier Roussel and Vasco Manquinho, *Input/Output Format and Solver Requirements for the Competitions of Pseudo-Boolean Solvers*, `http://www.cril.univ-artois.fr/PB16/format.pdf`, 2016.