General OPB Format

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 :

- separate the description of the general OPB format from the description of the restricted format used in the competition
- allow both min and max keywords in the objective function
- allow any usual relational operator in constraints (including $\neq$)
- allow any identifier for variable names
- allow to use Unicode characters and the UTF-8 encoding (for relational operators and in quoted variable names)
- extend the OPB format to support model counting and model enumeration (see Section 3.3).

1 Introduction

This document presents the general OPB format, which is used to represent pseudo-Boolean constraints and different problems based on these constraints. A high quality pseudo-Boolean solver is expected to read this format in order to support any instance written by a user. However, the competitions of pseudo-Boolean solvers use a restricted version of this format to simplify the implementation of a solver. This restricted format is described in a separate document.

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 four different problems: PBS (Pseudo-Boolean Satisfaction), PBO (Pseudo-Boolean Optimization), WBO (Weighted-Boolean Optimization) and model counting or enumeration. The OPB format allows to express constraints with integer coefficients of arbitrary size.

2 Input Format

This section details the input format for

- Pseudo-Boolean Satisfaction (PBS) instances (Section 3.1)
  the solver must find an interpretation which satisfies each constraint (decision problem)
- Pseudo-Boolean Optimization (PBO) instances (Section 3.1)
  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 3.2)
  the solver must find an interpretation which minimizes the cost of violated constraints (optimization problem)
- Model counting or enumeration instances (Section 3.3)
  the solver must count or enumerate the models of a pseudo-Boolean formula

3 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.

3.1 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:` or `max:`) 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.

An OPB file can be written in pure ASCII, or can be encoded in UTF-8 to use Unicode characters to represent the relational operators or use variable names in any language (between double quotes). Since most characters in the file are ASCII characters, it does not make sense to use UTF-16 or UTF-32. Besides, this would require significant changes in the existing parsers.

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:"|"max:"

<relational_operator>::=
    "=" | "!=" | "≠" | ">" | ">=" | "≥" | "<" | "<=" | "≤"

<variableName>::=
    <asciiLetterOrUnderscore>
    | <asciiLetterOrUnderscore> <sequenceOfAsciiLettersOrDigitsOrUnderscores>
    | <quotedString>

<quotedString>::= '"' <sequenceOfAnyCharacterExceptDoubleQuote> '"'

<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.

Variables names may follow the usual definition (start with a letter or underscore, followed by any number of letters, digits or underscore) or use an extended syntax (start with a double quote, followed by any non empty sequence of any character except double quote, end with a double quote) to allow using almost any notation in virtually any language. As it does not seem very useful to allow a variable name to contain a double quote, no escape sequence is defined and it may not appear in the name, which simplifies parsing.

Integers may be of arbitrary size (i.e. contain any number of digits) (see section 3).

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 or maximize. It starts with the word "min:" or "max:" followed by the linear function to minimize or maximize 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.

- The weight of a variable may contain an arbitrary number of 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.

3.1.1 Linear instances

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

```
<term>::=<literal>

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

The symbol $\sim$ is used to denote the negation of a variable.

Examples

```
*
* 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;
```

3.1.2 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>
```

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

```
*
* 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;
12345678901234567890 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.

```
* 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;
```

3.2 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 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.

3.2.1 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 ;
```

3.3 Model counting and model enumeration

Model counting consists in computing the number of models of a pseudo-Boolean formula. The solver has to count this number of models when the first non comment line of the file is equal to:

```
models: count ;
```

Model enumeration consists in listing all the models of a pseudo-Boolean formula. The solver has to enumerate these models when the first non comment line of the file is equal to:

```
models: enumerate ;
```

By default, model counting or enumeration is done on the whole vocabulary of the formula. Two models differ when any of the variables occurring in the formula has a different value.

Projection consists in considering only a subset P of relevant variables. Two models differ when any of the variables occurring in P has a different value. Models that assign the same value to variables in P but only differ on a variable not present in P are equivalent, and therefore ignored in the enumeration or the counting process.

To request projected model counting or enumeration, the second non comment line of the file will start with the keyword "project:" followed by the list of relevant variables (members of P). The list and the line is ended by a semicolon. As an example, the line below requests projection of the three variables $x1, x2$ and $x3$.

```
project: x1 x2 x3 ;
```

ALTERNATE REPRESENTATION

A simplified, one line specification of the model counting/enumeration could be chosen.
For model counting, the first non comment line would start with "count:" followed by the list of relevant variables (projection) followed by a semicolon. The list is empty when all variables must be considered. Two examples are given below:

```
* request model counting on all variables
count: ;

* request model counting on x1, x3 and x3
count: x1 x2 x3 ;
```

For model enumeration, the first non comment line would start with "enum:" followed by the list of relevant variables (projection) followed by a semicolon. The list is empty when all variables must be considered. Two examples are given below:

```
* request model emumeration on all variables
enum: ;

* request model enumeration on x1, x3 and x3
enum: x1 x2 x3 ;
```

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.