Satisfiability
Suggested Format

Last revision: May 8, 1993

This paper outlines a suggested format for satisfiability problems. It is not yet the "official" DIMACS graph format. If you have comments on this or other formats or you have information you think should be included, please send a note to `challenge@dimacs.rutgers.edu`.

1 Introduction

One purpose of the DIMACS Challenge is to ease the effort required to test and compare algorithms and heuristics by providing a common testbed of instances and analysis tools. To facilitate this effort, a standard format must be chosen for the problems addressed. This document outlines two formats for satisfiability problems. The purpose of these formats is to allow quick conversion from one format to another while still being reasonably effective formats directly.

Two formats were chosen to reflect the need for both a specialized format for satisfiability problems in conjunctive normal form and for a general format able to handle all types of satisfiability problems. These formats will be referred to as CNF format and SAT format respectively.

2 File Formats for Satisfiability Problems

This section describes a standard file format for graph inputs and outputs. There is no requirement that participants follow these specifications; however, compatible implementations will be able to make full use of DIMACS support tools.

Participants are welcome to develop translation programs to convert instances to and from more convenient, or more compact, representations; the Unix **awk** facility is recommended as especially suitable for this task.

All files contain ASCII characters. Input and output files contain several types of *lines*, described below. A line is terminated with an end-of-line character. Fields in each line are separated by at least one blank space.

2.1 CNF format

A satisfiability problem in conjunctive normal form consists of a the conjunction of a number of *clauses*, where is clause is a disjunction of a number of variables or their negations. If we let x_i represent variables that can assume only the values *true* or *false*, then a sample formula in conjunctive normal form would be

$$(x_1 \vee x_3 \vee \bar{x_4}) \wedge (x_4) \wedge (x_2 \vee \bar{x_3})$$

where $\vee$ represents the *or* boolean connective, $\wedge$ represents *and* and $\bar{x}_i$ is the negation of x_i.

Given a set of clauses $C_1, C_2, \ldots, C_m$ on the variables $x_1, x_2, \ldots, x_n$, the satisfiability problem is to determine if the formula

$$C_1 \wedge C_2 \wedge \ldots \wedge C_m$$

is satisfiable. That is, is there an assignment of values to the variables so that the above formula evaluates to *true*. Clearly, this requires that each C_j evaluate to *true*.

The maximum satisfiability problem is to find an assignment of values to the variables so as to have the maximum number of C_j evaluate to *true*.

To represent an instance of such problems, we will create an input file that contains all of the information needed to define a satisfiability problem or a maximum satisfiability problem. This file will be an ASCII file consisting of a two major sections: the preamble and the clauses.

The Preamble. The preamble contains information about the instance. This information is contained in lines. Each line begins with a single character (followed by a space) that determines the type of line. These types are as follows:

- **Comments.** Comment lines give human-readable information about the file and are ignored by programs. Comment lines appear at the beginning of the preamble. Each comment line begins with a lower-case character **c**.

  ```
  c This is an example of a comment line.
  ```

- **Problem line.** There is one problem line per input file. The problem line must appear before any node or arc descriptor lines. For cnf instances, the problem line has the following format.

  ```
  p FORMAT VARIABLES CLAUSES
  ```

  The lower-case character `p` signifies that this is the problem line. The `FORMAT` field allows programs to determine the format that will be expected, and should contain the word "cnf". The `VARIABLES` field contains an integer value specifying n, the number of variables in the instance. The `CLAUSES` field contains an integer value specifying m, the number of clauses in the instance. This line must occur as the last line of the preamble.

The Clauses. The clauses appear immediately after the problem line. The variables are assumed to be numbered from 1 up to n. It is not necessary that every variable appear in an instance. Each clause will be represented by a sequence of numbers, each separated by either a space, a tab, or a newline character. The non–negated version of a variable i is represented by i; the negated version is represented by $-i$.

Each clauses is terminated by the value 0. Unlike many formats that represent the end of a clause by a new–line character, this format allows clauses to be on multiple lines.

Example. Using the example

$$(x_1 \vee x_3 \vee \bar{x_4}) \wedge (x_4) \wedge (x_2 \vee \bar{x_3})$$

a possible input file would be

```
c Example CNF format file

c

p cnf 4 3

1 3 -4 0

4 0 2

-3
```

2.2 SAT format

Conjunctive normal form is not the only natural encoding for satisfiability problems. There are other encodings that lead to interesting satisfiability problems but whose translation into CNF unnecessarily increases the size of the problem. To allow formulation of such instances, as well as providing an alternative form for CNF format, the following format is also supported. This file consists also of a preamble and a formula section.

The Preamble. The preamble contains information about the instance. This information is contained in lines. Each line begins with a single character (followed by a space) that determines the type of line. These types are as follows:

- **Comments.** Comment lines give human-readable information about the file and are ignored by programs. Comment lines appear at the beginning of the preamble. Each comment line begins with a lower-case character **c**.

```
c This is an example of a comment line.
```

- **Problem line.** There is one problem line per input file. The problem line must appear before any node or arc descriptor lines. For network instances, the problem line has the following format.

  ```
  p FORMAT VARIABLES
  ```

  The lower-case character `p` signifies that this is the problem line. The `FORMAT` field allows programs to determine the format that will be expected, and should contain the word "sat". The `VARIABLES` field contains an integer value specifying n, the number of variables in the instance. This line must occur as the last line of the preamble.

The Formula. Immediately after the problem statement, the formula appears. This formula consists of one or more lines, containing the formula to be satisfied. The variables are represented by the numbers 1 through n. Negation of a variable i is represented by $-i$. Valid formulae are represented by the following rules:

1. i and $-i$ are formula for all i.
2. If f is a valid formula, so is (f).
3. If f is a valid formula, so is $-(f)$.
4. If $f_1, f_2, \ldots, f_k$ are valid formulas, so is $*(f_1\ f_2\ \ldots f_k)$.
5. If $f_1, f_2, \ldots, f_k$ are valid formulas, so is $+(f_1\ f_2\ \ldots f_k)$.

White space separating pieces of a formula can either be spaces, tabs, or newline characters. Whitespace is not required where the tokens are unambiguous without it. In particular, both (1 –2) and (1-2) are valid formulae. $*()$ and $+()$ are valid and interpreted as TRUE and FALSE respectively.

The "*" operator represents the *and* operation, the "+" represents the *or* operation, and "$-$" represents negation.

The formula represented must be of the form (f), for a valid formula f.

Example. For the formula

$$(x_1 \vee x_3 \vee \bar{x_4}) \wedge (x_4) \wedge (x_2 \vee \bar{x_3})$$

a sample input file is

```
c Sample SAT format

c

p sat 4

(*(+(1 3 -4)

   +(4)

   +(2 3)))
```

2.3 Additions and Expansions

The purpose of the standard format is to have a common language for expressing problems. It may be that the formats chosen are not rich enough for some types of problems. If you would like to suggest any expansions, please contact the Challenge.

The following extensions have been defined:

XOR Format. Problem type is satx. New operator "xor" is defined with the same syntax as "+" OR "*". xor$(f_1 f_2 \ldots f_n)$ evaluates to true if and only if an odd number of $f_1, f_2, \ldots, f_n$ evaluate to true. xor() evaluates to false.

EQUAL Format. Problem type is sate. New operator "=" with syntax like "+" or "*". $= (f_1 f_2 \ldots f_n)$ evaluates to true if and only if $f_1, f_2, \ldots, f_n$ are either all true or all false. $= ()$ evaluates to true.

XOR–EQUAL Format. Combines the above with problem type satex.

2.4 Output Files

Every algorithm or heuristic should create an output file. This output file should consist of one or more of the following lines, depending on the type of algorithm and problem being solved.

- **Comments.** Comment lines give human-readable information about the file and are ignored by programs. Comment lines can appear anywhere in the file. Each comment line begins with a lower-case character `c`. Note that comment lines can be used to provide solution information not otherwise available (i.e. computation time, number of calculations).

  ```
  c This is an example of a comment line.
  ```

- **Solution Line**

  ```
  s TYPE SOLUTION VARIABLES CLAUSES
  ```

  ```
  s TYPE SOLUTION VARIABLES
  ```

  The lower-case character `s` signifies that this is a solution line. The `TYPE` field denotes the type of solution contained in the file. This should be one of the following strings: "max", for solving the maximum satisfiability problem (whose input file format was necessarily "cnf"), or the FORMAT string from the *Problem Line*, for solving (some form of) the satisfiability problem. In particular, for CNF satisfiability, the string is "cnf". See *Problem Line* description for other possibilities.

  The `SOLUTION` field contains an integer corresponding to the solution value. For maximum satisfiability, this should be the number of clauses satisfied; for satisfiability, this should be "1" if the formula is satisfiable, 0 if the formula is unsatisfiable, and -1 if no decision was reached.

The `VARIABLES` field contains the same integer that was in the VARIABLES field of the problem line. The `CLAUSES` field contains the same integer that was in the CLAUSES field of the problem line, applicable to "cnf" format only.

Notice that a Solution Line "of last resort" can be appended to the output file by a Unix shell script in which the program is executing, in the event that the program dies prematurely.

- **Timing Line**

  The Timing Line is optional, but is recommended. Its purpose is to standardize the reporting of timing information for ease of statistical analysis. It may appear anywhere in the file, and it repeats all of the information on the solution line for simplicity of extraction.

  ```
  t TYPE SOLUTION VARIABLES CLAUSES CPUSECS MEASURE1 ...
  ```

  The lower-case character `t` signifies that this is a timing line. The `TYPE`, `SOLUTION`, `VARIABLES`, and `CLAUSES` fields are identical to the solution line, except that the `CLAUSES` field is 0 when not applicable.

  The `CPUSECS` field is a floating point number designating the number of CPU seconds used during the solution (or attempted solution). All numbers should be understandable by `awk`. A number without a decimal point is acceptable as "floating point".

  Remaining fields are floating point numbers providing alternative measures of performance that are "algorithmic" and reproducible: that is, these numbers should come out the same under different system loads, and on different architectures. `MEASURE1` is required (just print 0 to abstain), and is what the application thinks is the most significant measure of performance, such as number of nodes in search space, number of variable-assignment changes, etc.

  Additional measures report other interesting performance data, depending on the application. Exceptions to the rule of reproducibility may be made for additional measures: for example, memory requirement might vary by architecture.

Notice that a Timing Line "of last resort" can be appended to the output file by a Unix shell script in which the program is executing.

- **Variable Line**

  ```
  v V
  ```

  The lower-case character `v` signifies that this is a variable line. The value V is either a positive value i, which means that i should be set *true* or a negative value $-i$, implying it should be set false.

- **Clause Satisfaction Line**

  ```
  s C
  ```

  This line, useful only for maximum satisfiability, denotes whether a particular clause is satisfied or not. The lower-case character `s` signifies that this is a clause line. The value C is either a positive value i, which means that clause i is satisified by the solution, or a negative value $-i$, implying it is not.

3 Implementation at DIMACS.

CNF format files will generally have a `.cnf` extension, while SAT format files will have a `.sat` extension.