# Detecting Cardinality Constraints in CNF

Armin Biere[1], Daniel Le Berre[2], Emmanuel Lonca[2], and Norbert Manthey[3]

[1] Johannes Kepler University
[2] CNRS Université d'Artois
[3] Technische Universität Dresden

**Abstract.** We present novel approaches to detect cardinality constraints expressed in CNF. The first approach is based on a syntactic analysis of specific data structures used in SAT solvers to represent binary and ternary clauses, whereas the second approach is based on a semantic analysis by unit propagation. The syntactic approach computes an approximation of the cardinality constraints AtMost-1 and AtMost-2 constraints very fast, whereas the semantic approach has the property to be generic, i.e. it can detect cardinality constraints AtMost-$k$ for any $k$, at a higher computation cost. Our experimental results suggest that both approaches are efficient at recovering AtMost-1 and AtMost-2 cardinality constraints.

## 1 Introduction

Current benchmarks in CNF contain various Boolean functions encoded with clauses [30,15]. Among them, cardinality constraints $\sum_{i=1}^{n} l_i \otimes k$ with $\otimes \in \{<, \leq, =, \geq, >\}$ are Boolean functions whose satisfiability is determined by counting the satisfied literals on the left hand side and compare them to the right hand side (the *threshold*). For instance, $x_1 + x_2 + \neg x_3 + \neg x_4 \leq 2$ is satisfied iff at most 2 of its literals are satisfied. A wide use case of those constraints is to encode that a domain variable $v$ takes one value of the discrete set $\{o_1, o_2, \dots, o_n\}$, which is represented by the $n$ Boolean variables $v_{o_i}$ and the cardinality constraint $\sum v_{o_i} = 1$.

Since cardinality constraints are Boolean functions, they can be expressed by an equivalent CNF. The "theoretical" approach, i.e. the one found in [12] for instance, translates a cardinality constraint $\sum_{i=1}^{n} l_i \leq k$ using $\binom{n}{k+1}$ negative clauses of size $k+1$. Such encoding is called *binomial* because of the number of generated clauses. In practice, introducing new variables to reduce the number of clauses in the CNF usually results in a better performance. Various encodings have been proposed in the last decade (see for instance [14] for a survey). We discuss commonly used encodings in next section.

Pseudo-Boolean solvers use a proof system like *generalized resolution* [21], which is a specific form of the *cutting planes* proof system [12] that *p-simulates* resolution. This way, these solvers are able to solve instances of the Pigeon Hole Principle [19] when they are given cardinality constraints but not when they are given the same problem expressed with clauses. The reason of that behavior is that applying generalized resolution on clauses is equivalent to resolution [21], while on cardinality constraints generalized resolution is a specific form of cutting planes [12]. Retrieving cardinality constraints from clauses in the cutting planes proof system requires a very specific procedure. Take for instance the cardinality constraint

$$x_1 + x_2 + x_3 + x_4 \leq 1$$

which is equivalent to

$$\overline{x_1} + \overline{x_2} + \overline{x_3} + \overline{x_4} \geq 3$$

This cardinality constraint is represented in CNF using the following clauses:

$$\neg x_1 \vee \neg x_2, \quad \neg x_1 \vee \neg x_3, \quad \neg x_1 \vee \neg x_4, \quad \neg x_2 \vee \neg x_3, \quad \neg x_2 \vee \neg x_4, \quad \neg x_3 \vee \neg x_4$$

These clauses can be represented as binary cardinality constraints:

$$x_1 + x_2 \leq 1, \quad x_1 + x_3 \leq 1, \quad x_1 + x_4 \leq 1, \quad x_2 + x_3 \leq 1, \quad x_2 + x_4 \leq 1, \quad x_3 + x_4 \leq 1$$

Retrieving the original cardinality from the clauses represented by cardinalities $\leq 1$ requires to derive intermediate constraints as shown below (from [12]):

$$\frac{\begin{array}{c} x_1 + x_2 \leq 1 \\ x_1 + x_3 \leq 1 \\ x_2 + x_3 \leq 1 \end{array}}{x_1 + x_2 + x_3 \leq 1} \quad \frac{\begin{array}{c} x_1 + x_2 \leq 1 \\ x_1 + x_4 \leq 1 \\ x_2 + x_4 \leq 1 \end{array}}{x_1 + x_2 + x_4 \leq 1} \quad \frac{\begin{array}{c} x_1 + x_3 \leq 1 \\ x_1 + x_4 \leq 1 \\ x_3 + x_4 \leq 1 \end{array}}{x_1 + x_3 + x_4 \leq 1} \quad \frac{\begin{array}{c} x_2 + x_3 \leq 1 \\ x_2 + x_4 \leq 1 \\ x_3 + x_4 \leq 1 \end{array}}{x_2 + x_3 + x_4 \leq 1}$$

For the first column, summing the three cardinality constraints leads to $2x_1 + 2x_2 + 2x_3 \leq 3$, which can be reduced to $x_1 + x_2 + x_3 \leq 1$ by dividing the inequality by 2 and rounding down the threshold. The same process can be applied to derive the other cardinality constraints in the last line. Finally, summing up these four cardinality constraints of 3 literals results in a cardinality constraint of 4 literals: $3x_1 + 3x_2 + 3x_3 + 3x_4 \leq 4$. The expected cardinality constraint $x_1 + x_2 + x_3 + x_4 \leq \lfloor \frac{4}{3} \rfloor$ is obtained after division by 3 and rounding.

The described process is tedious and not easy to integrate in a solver. Thus, the idea is to find a way to detect those cardinality constraints in a preprocessing step, independent from the original proof system of the solver.

The motivation for this work is to allow solvers to take advantage of those cardinality constraints, at least for space efficiency (support of native cardinality constraints) or because of a better proof system (e.g. Generalized Resolution [21] or Cutting Planes [12]). Detecting cardinality constraints is also an interesting idea for pure SAT solvers, namely for constraints reencoding, e.g. to encode cardinality constraints back to CNF with an alternative and hopefully more efficient encoding [27,26]. This is especially useful in practice to replace the commonly used *pairwise* encoding of $\leq 1$ constraints with a more efficient encoding.

## 2 Short Review of Known Encodings

Before we discuss how to find encoded cardinality constraints, a few common encodings for widely used constraints are introduced. For the AtMost-1 constraint