Extended Conjunctive Normal Form and
An Efficient Algorithm for Cardinality Constraints

Zhendong Lei¹,², Shaowei Cai¹,²* and Chuan Luo³

¹State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
²School of Computer Science and Technology, University of Chinese Academy of Sciences, China
³Microsoft Research, China
{leizd, caisw}@ios.ac.cn

Abstract

Satisfiability (SAT) and Maximum Satisfiability (MaxSAT) are two basic and important constraint problems with many important applications. SAT and MaxSAT are expressed in CNF, which is difficult to deal with cardinality constraints. In this paper, we introduce Extended Conjunctive Normal Form (ECNF), which expresses cardinality constraints straightforward and does not need auxiliary variables or clauses. Then, we develop a simple and efficient local search solver LS-ECNF with a well designed scoring function under ECNF. We also develop a generalized Unit Propagation (UP) based algorithm to generate the initial solution for local search. We encode instances from Nurse Rostering and Discrete Tomography Problems into CNF with three different cardinality constraint encodings and ECNF respectively. Experimental results show that LS-ECNF has much better performance than state of the art MaxSAT, SAT, Pseudo-Boolean and ILP solvers, which indicates solving cardinality constraints with ECNF is promising.

1 Introduction

Satisfiability problem (SAT) has gained a considerable attention with the advent of a new generation of solvers able to solve large instances from real world problems. Maximum Satisfiability (MaxSAT) is an optimization version of SAT which also achieves great success thanks to the techniques used in SAT. These impressive progress led to more and more combinatorial optimization problems in real world situations encoded into these propositional logic languages and solved by applying a suitable SAT or MaxSAT solver.

Given a propositional formula expressed in the Conjunctive Normal Form (CNF), SAT is concerned with satisfying all clauses, while MaxSAT is concerned with satisfying as many clauses as possible. Partial MaxSAT (PMS) is a generalization of MaxSAT, whose clauses are divided into hard and weighted soft clauses and the goal is to find an assignment that satisfies all the hard clauses and maximize the total weight of satisfied soft clauses. There are many efficient PMS solvers, including SAT-based solvers [Narodytska and Bacchus, 2014; Martins et al., 2014; Berg et al., 2019; Nadel, 2019; Joshi et al., 2019; Demirovic and Stuckey, 2019] and local search solvers [Cai et al., 2016; Luo et al., 2017; Cai et al., 2017; Lei and Cai, 2018; Guerreiro et al., 2019].

One of the most important drawbacks of these logical languages is the difficulty to deal with cardinality constraints. Indeed, cardinality constraints arise frequently in the encoding of many real world situations such as scheduling, logic synthesis or verification, product configuration and data mining. For the above reasons, many works have been done on finding an efficient encoding of cardinality constraints in CNF formulas [Sinz, 2005; Asín et al., 2009; Hattad et al., 2017; Boudane et al., 2018; Karpinski and Piotrów, 2019].

Even though modern SAT and MaxSAT solvers are powerful and the cardinality constraint encodings have also been improved, MaxSAT and SAT solvers still do not perform well when solving problems with cardinality constraints. Many encoding methods have been proposed, but their performance vary among different domains. Indeed, experiment results from a few problems [Demirovic and Musliu, 2014; Demirovic et al., 2019] indicate that none of these encoding methods are obviously dominant.

In this work, we propose another solution to deal with cardinality constraints. Instead of finding a better encoding method, we propose Extended Conjunctive Normal Form (ECNF), which is an extension of CNF. In SAT(MaxSAT) of CNF, a clause is satisfied if it contains at least one true literal, while in SAT(MaxSAT) of ECNF, a clause is satisfied if it contains a certain amount of true literals. Encoding cardinality constraints into ECNF is natural and simple without introducing any additional variable or clause. The most important thing is that techniques used in SAT or MaxSAT solvers can be easily adapted to this model with minor modifications.

We define the ECNF versions of SAT and MaxSAT problems. Then we develop a local search solver LS-ECNF for solving SAT and MaxSAT of ECNF formulas. A new scoring function is used in LS-ECNF to adapt to the new model. We also design a generalized Unit Propagation (UP) based algorithm to get an initial solution for LS-ECNF. To study the effectiveness of our method, we encode instances from Nurse Rostering and Discrete Tomography Problems into MaxSAT(SAT) of CNF with three different cardinal-
ity constraint encodings respectively. We also encode these instances into ECNF for comparison. Experimental results show that LS-ECNF has great improvement in the encoding efficiency, memory usage, speed and quality over the state of the art MaxSAT, SAT, Pseudo-Boolean and even Integer Linear Programming (ILP) solvers.

The remainder of this paper is structured as follows. Section 2 introduces preliminary knowledge. Section 3 presents the definition of ECNF and encoding rules from cardinality constraints to ECNF. Section 4 presents the local search and generalized UP-based algorithms. Experimental results are presented in Section 5. Finally, we conclude the paper.

2 Preliminary

Given a set of \( n \) Boolean variables \( \{x_1, x_2, ..., x_m\} \), a literal is either a variable \( x_i \) or its negation \( \neg x_i \). A clause is a disjunction of literals which can be represented as a set of literals. A conjunctive normal form (CNF) formula \( F = C_1 \land C_2 \land ... \land C_m \) is a conjunction of clauses which can be represented as a multiset of clauses. A complete truth assignment is a mapping that assigns to each variable either 0 or 1.

In CNF, a clause is satisfied if it has at least one true literal, and falsified otherwise. SAT problem is concerned with satisfying all clauses, while MaxSAT problem is concerned with satisfying as many clauses as possible. Given a CNF formula, the Partial MaxSAT (PMS) problem, in which some clauses are declared to be hard and the rest are declared to be soft, is the problem of finding an assignment such that all hard clauses are satisfied and maximizes the total number of satisfied soft clauses. In Weighted PMS (WPMS), each soft clause is associated with a positive integer as its weight, and the goal is to satisfy all hard clauses and maximize the total weight of satisfied soft clauses.

Note that, if the problem contains only hard (resp. soft) constraints, the obtained problem is SAT (resp. MaxSAT). If the problem contains both hard and soft constraints, the resulting problem is Partial MaxSAT.

A unit clause is a clause containing only one literal. The unit propagation technique is quite simple: For a given CNF formula, we collect all unit clauses in it, and then assume that variables are set to satisfy these unit clauses. That is, if the unit clause \( \{v_i\} \) appears in the formula, we set \( v_i \) to true; and if the unit clause \( \{-v_i\} \) appears in the formula, we set \( v_i \) to false. We then condition the formula on these settings. The iterative application of this rule until no more unit clauses remain is called unit propagation (UP).

2.1 Cardinality Constraints

Cardinality constraints impose limits on the truth values assigned to literals. These are \( \text{atLeast}_k[x_i : x_i \in X] \), \( \text{atMost}_k[x_i : x_i \in X] \) and \( \text{exactly}_k[x_i : x_i \in X] \), which constraint that at least, at most and exactly \( k \) literals out of the specified ones must or may be assigned to true. For clarity and consistency, and as the constraints \( \text{atMost}_k[x_i : x_i \in X] \) can be equivalently rewritten as an \( \text{atLeast}_{l}(m - k)[\neg x_i : x_i \in X] \) and \( \text{exactly}_k[x_i : x_i \in X] \) can be rewritten as \( \text{atLeast}_k[x_i : x_i \in X] \) and \( \text{atLeast}_{l}(m - k)[\neg x_i : x_i \in X] \), we only consider the \( \text{atLeast}_k \) constraints. Usually, each cardinality constraint \( c \) has a penalty multiplier \( \lambda(c) \).

We differentiate hard and soft cardinality constraints. Hard cardinality constraints are the ones that must be satisfied. Soft cardinality constraints are the ones that can be falsified with a penalty to the cost. In our case, the penalty is proportional to the severity of the violation. Specifically, the penalty is the product of the multiplier \( \lambda \) and the degree of violation (which is the minimum number of variables in the clause that need to be flipped in order to meet the cardinality constraint). For example, for the soft constraints \( \text{atLeast}_3[x_1, x_2, x_3, x_4] \) with penalty multiplier \( \lambda \), the assignment \((1, 1, 0, 0)\) would incur no penalty, while assignment \((1, 1, 0, 0)\) and \((0, 1, 0, 0)\) would incur a penalty of \( 1 \lambda \) and \( 2 \lambda \) respectively.

2.2 Encoding Cardinality Constraints into CNF

Different methods of encoding cardinality constraints into CNF have been studied. We choose three most famous encoding methods in recent years: cardinality networks, sequential encoding and pigeon-hole encoding.

**Sequential encoding:** The main idea behind the sequential encoding is to encode the sum of the considered variables and then forbid certain output values. It needs \( O(k \ast m) \) auxiliary clauses and variables where \( k \) is the cardinality of the constraint and \( m \) is the number of the variables of the constraint. Generalized arc consistency is maintained by unit propagation. For more details, we refer the reader to [Boudane et al., 2018; Sinz, 2005].

**Cardinality networks encoding:** Cardinality networks generate helper variables that are used to sort all the considered truth assignments and then insert clauses which forbid certain outputs. Generalized arc consistency is maintained by unit propagation. It needs \( O(m \ast \log_2(k)) \) auxiliary clauses and variables. For more details on the encoding, we refer the reader to [Boudane et al., 2018; Asín et al., 2009].

**Pigeon-hole encoding:** The semantic of the cardinality constraint can be equivalently expressed as the problem of putting \( k \) pigeons into \( n \) holes. We use the improved version obtained by breaking the symmetries and this version also maintains generalized arc consistency. It needs \( O(k \ast (m - k)) \) auxiliary clauses and variables. More details can be found in [Boudane et al., 2018; Hattad et al., 2017].

These encodings are proposed to encode hard cardinality constraints into SAT. Actually, they can also be applied to encode soft cardinality constraints. In [Demirovic and Musliu, 2014; Demirovic et al., 2019], they showed how to encoding soft constraints into MaxSAT and we take the same methods. The number of auxiliary variables and clauses used for encoding a soft cardinality constraint is almost the same as the ones used for a hard cardinality constraint.

3 Extended Conjunctive Normal Form

In this section, we introduce Extended Conjunctive Normal Form. We first give the definitions and notations below.

**Extended Conjunctive Normal Form.** ECNF is extended from CNF as follows. (1) In ECNF, each clause is associated with a positive integer cardinality\([c]\) as its cardinality. (2)
In ECNF, all cardinality constraints are expressed in the form of \( \text{atLeast}_k \) constraints. (3) A clause is satisfied if it has at least \( \text{cardinality}[c] \) true literals. (4) In addition, as with CNF, we differentiate hard and soft clauses. Each soft clause \( s \) is associated with an integer as its penalty multiplier \( \lambda[s] \).

**SAT of ENCF formulas.** Given an ECNF formula, the definition of SAT remains the same, i.e., to decide whether all clauses can be satisfied, and provide an satisfying assignment if the answer yes.

**PMS of ENCF formulas.** Given a ECNF formula and an assignment \( \alpha \) to its variables, let \( \text{num\_true}_I[s] \) denote the number of true literals in clause \( s \), the penalty of a falsified soft clause \( s \) is defined as
\[
\text{penalty}[s] = \lambda[s] \cdot (\text{cardinality}[s] - \text{num\_true}_I[s]) .
\]
Note that we adopt penalty of the linear form, which is adopted in most works dealing with cardinality constraints. But the penalty can be defined using other forms of functions, according to the original problems.

The PMS problem of ECNF formulas, is to find an assignment to satisfy all hard clauses, and to minimize the total penalty of falsified clauses. An assignment \( \alpha \) is feasible if all hard clauses are satisfied, and the cost of \( \alpha \), denoted by \( gcost(\alpha) \), is the total penalty of falsified soft clauses under \( \alpha \).

For convenience, we denote SAT and Partial MaxSAT problems for ECNF formulas as ESAT and EPMS respectively. If all cardinality of clauses in an ECNF formula is 1, it becomes a CNF formula.

**Encoding method.** We describe our method of encoding cardinality constraints into ECNF as follows. In our method, we rewrite \( \text{atMost}_k \) and \( \text{exactly}_k \) constraints as \( \text{atLeast}_k \) constraints (described in Section 2.1), and thus we only need to consider the encoding of \( \text{atLeast}_k \) constraints.

- For a hard cardinality constraint \( \text{atLeast}_k[x_1 : x_i \in X] \), generate a hard clause \( c := \{x_i : x_i \in X\} \) with \( \text{cardinality}[c] = k \).
- For a soft cardinality constraint \( \text{atLeast}_k[x_i : x_i \in X] \) and the penalty multiplier \( \lambda \), generate a soft clause \( s := \{x_i : x_i \in X\} \) with \( \text{cardinality}[s] = k \) and \( \lambda[s] = \lambda \).

4 Local Search Algorithm for EPMS

We introduce a simple local search algorithm \( LS-ECNF \) and then propose a generalized unit propagation based decimation algorithm to generate the initial solution. Note that, \( LS-ECNF \) can be used to solve both ESAT and EPMS problems.

Clause weighting techniques are very common in local search algorithms for SAT and MaxSAT. We propose a new weighting scheme for EPMS and the weight maintained by clause weighting techniques is denoted by \( w(c) \).

The clause weighting scheme is named as Weighting-EPMS, and works as follows: For each hard clause \( c \), we associate an integer number as its weight which is initialized to 1; for each soft clause \( s \), we use \( \lambda[s] \) as its initial weight. Whenever a “stuck” situation is observed, that is, we cannot decrease the cost by flipping any variable, the clause weights are updated as follows.

\[ w(c) := \begin{cases} 1 & \text{if } c \text{ is falsified hard clause} \\ w(c)+1 & \text{if } w(c)<\zeta (\zeta = 1000) \end{cases} \]

- with probability \( 1-sp \): for each falsified hard clause \( c \), \( w(c) := w(c)+1 \); for each falsified soft clause \( c \), \( w(c) := w(c)+1 \) if \( w(c)<\zeta \).
- with probability \( sp \) (smoothing probability): for each satisfied clause \( c \), \( w(c) := w(c)-1 \).

Local search algorithms use scoring functions and heuristics to guide the search. The most common used scoring function is perhaps the score of variables. In local search for MaxSAT, the score of a variable \( x \) is the increase of total weight of satisfied clauses caused by flipping \( x \). In EPMS, the penalty differs according to the severity of the violation, so the traditional scoring functions in local search for MaxSAT are not suitable for EPMS.

**Definition 1. (Penalty weight of falsified clauses)** For each clause \( c \) in EPMS (either hard or soft clause), if \( \text{num\_true}_I[c] < \text{cardinality}[c] \), it incurs a penalty of \( w(c) \cdot (\text{cardinality}[c] - \text{num\_true}_I[c]) \).

**Definition 2. (Score in LS-ECNF)** In LS-ECNF, the score of a variable \( x \), denoted by \( \text{score}(x) \), is the decrease of total penalty weight of falsified clauses caused by flipping \( x \).

Here is an example. Consider a hard clause \( c = \{x_1, x_2, x_3, x_4, x_5\} \) with \( \text{cardinality}[c] = 3 \), and suppose its current weight is 2, given the assignment \( \{x_1, x_2, x_3, x_4, x_5\} = (1, 0, 0, 0, 0) \), \( \text{score}(x_2) = 2 \) and \( \text{score}(x_1) = -2 \) according to our scoring function in \( LS-ECNF \). For another assignment \( (1,0,1,1,0) \), \( \text{score}(x_2) = 0 \) as the number of true literals \( \text{num\_true}_I[c] = \text{cardinality}[c] \).

4.1 The \( LS-ECNF \) Algorithm

Our local search algorithm for EPMS (Partial MaxSAT under ECNF), denoted by \( LS-ECNF \), is presented below. In the beginning, \( LS-ECNF \) generates a complete assignment \( \alpha \), and the best found solution \( \alpha^* \) is initialized as \( \emptyset \).

Algorithm 1: \( LS-ECNF \)

**Input:** EPMS instance \( F, \text{cutoff} \)

**Output:** A feasible assignment \( \alpha \) of \( F \) and its cost, or “no feasible assignment found”

\[
\begin{align*}
\text{begin} & \\
\alpha := \text{an initial complete assignment}; & \alpha^* := \emptyset; \\
\text{while} & \text{elapsed time} < \text{cutoff} \text{ do} \\
\quad & \text{if} \# \text{falsified hard clauses} \& \text{ and } \text{gcost}(\alpha) < \text{gcost}^* \\
\quad & \text{then} \quad \alpha^* := \alpha; \quad \text{gcost}^* := \text{gcost}(\alpha); \\
\quad & \text{if } D := \{x|\text{score}(x) > 0\} \neq \emptyset \text{ then} \\
\quad & \quad v := \text{a variable in } D \text{ with the highest score}; \\
\quad & \text{else} \\
\quad & \quad \text{update weights of clauses by Weighting-EPMS}; \\
\quad & \text{if } \exists \text{ falsified hard clauses} \text{ then} \\
\quad & \quad c := \text{a random falsified hard clause}; \\
\quad & \text{else} \\
\quad & \quad c := \text{a random falsified soft clause}; \\
\quad & \quad v := \text{the variable with highest score in } c; \\
\quad & \quad \alpha := \alpha \text{ with } v \text{ flipped}; \\
\quad & \text{if } \alpha^* \text{ is feasible then return } (\text{gcost}^*, \alpha^*); \\
\quad & \text{else return } \text{“no feasible assignment found”};
\end{align*}
\]
After that, a loop (lines 3-13) is executed to iteratively modify \( \alpha \) until a given time limit is reached. During the search, whenever a better feasible solution is found, the best feasible solution \( \alpha^* \), and \( g_{\text{cost}}^* \), are updated accordingly (line 4). At each iteration, if the set \( D \) of the variables with score bigger than zero is not empty, \( LS-ECNF \) picks the variable with the greatest score in \( D \) breaking ties by time stamp.

If there is no such variable, which means the search got stuck, \( LS-ECNF \) updates clause weights according to the Weighting-EPMS scheme, and picks a variable from a falsified clause. Specifically, it chooses a clause randomly from falsified hard clauses if any, and from falsified soft clauses otherwise. Then, \( LS-ECNF \) picks the variable with the greatest score in the clause and flips it.

Finally, when the loop terminates upon reaching the time limit, \( LS-ECNF \) returns \( g_{\text{cost}}^* \) and \( \alpha^* \) if \( \alpha^* \) is a feasible solution. Otherwise, it returns "no feasible assignment found".

4.2 UP-based Algorithm for EPMS

Unit Propagation (UP) has been used in SAT and MaxSAT solvers. Actually, UP can also be applied to EPMS with some modifications. Following the line of UP-based decimation [Cai et al., 2017], we integrate UP to design a decimation algorithm for EPMS, and use it to produce initial assignments for \( LS-ECNF \).

**Definition 3. Generalized Unit Clause** For a hard (resp. soft) clause \( c \), if its cardinality is bigger or equal than the number of literals in \( c \), then \( c \) is a generalized hard (resp. soft) unit clause.

**Generalized Unit Propagation:** For a given ECNF formula, we collect all generalized unit clauses in it, and all the literals in these generalized unit clauses are set to true. Then the formula is simplified accordingly.

The GUP-based decimation algorithm is outlined in Algorithm 2, and is described below. The algorithm works iteratively by assigning variables one by one (line 2-9) until no unassigned variables are left. In each step, it is divided into three situations. (1) Firstly, if there exist a generalized hard unit clause \( c \), the algorithm picks a hard unit clause randomly and performs a generalized unit propagation step using \( c \). (2) Secondly, if there is no generalized hard unit clause but soft ones, the algorithm picks a soft unit clause randomly and works in the same manner as in the first situation. (3) Lastly, if there is no generalized unit clause, then the algorithm picks an unassigned variable \( x \), assigns it randomly, and simplifies the formula accordingly.

5 Experimental Evaluation

We model Nurse Rostering and Tomography Problems into ECNF formulas. We also encode these two problems into CNF formula with three famous encoding types (sequential encoding, cardinality networks encoding and pigeon-hole encoding), Pseudo-Boolean (PB) and Integer Linear Programming (ILP) for comparison. Note that, we encode the original instances as PB or ILP instances directly which is fair for comparison. These two problems are described as follows.

**Nurse-Rostering Problem (NRP):** These are well-known problem instances from the second international nurse rostering competition (INRC2) [Ceschia et al., 2015]. We chose to focus on this specific problem formulation as it provides a number of instances that include challenging and realistic scheduling problems, while still being intuitive and straightforward to use. The overall goal is to find an optimal roster for a number of nurses, shift types, where every nurse may either work a single shift or have a day off on each day of a given scheduling period. Of course, that this arrangement should satisfy a set of constraints including both hard and soft ones. Finally, the objective function of a candidate solution is defined as the sum of penalty of all violated soft constraints. In this work, the penalty multiplier of each soft constraint is set to 1. The size of NRP instances varies according to the number of nurses and the length of scheduling period. In our experiments, the number of nurses ranges from 30 to 120 and the length of scheduling period ranges from 12 to 24 weeks.

**Discrete Tomography Problems:** This set of benchmarks we have used is the one introduced in [Bailleux and Boufkhad, 2003]. The idea is to first generate an \( N \times N \) grid in which some cells are filled and some others are not. The problem consists in finding out which are the filled cells there are in each row, column and diagonal. For that purpose, variables \( x_{ij} \) are used to indicate whether cell \((i, j)\) is filled. And cardinality constraints impose how many filled cells there are in each row, column and diagonal. We generate 170 benchmarks (10 instances for each size \( N = 10, 20, \ldots, 90, 100, 150, \ldots, 400 \)). Note that, the discrete tomography problem only contains hard cardinality constraints, and thus is a SAT problem.

We compare our algorithm against five state of the art algorithms including SAT, MaxSAT, Pseudo-Boolean and ILP solvers.

- **SATLike:** [Lei and Cai, 2018] is the best local search solver for WPMS and won two unweighted categories of incomplete track in MSE (MaxSAT Evaluation) 2018.
- **Loandra:** [Berg et al., 2019] won two unweighted categories, and was ranked 2nd in two weighted categories of incomplete track in MSE 2019.
for finding the feasible solution of each instance. For each cost of the best feasible solution found. Discrete tomography on nurse rostering problems. For nurse rostering problem, and the time limit is 3600 seconds, various time limits, including 300 seconds and 3600 seconds running Ubuntu 16.04.5 Linux operation system. We adopt using Intel Xeon E7-8850 v2 @2.30GHz, 2048GB RAM, with -O3 option. Our experiments were conducted on a server of the art solvers on nurse-rostering problem. Time limit is 300s

Table 1: Experiment results of comparing LS-ECNF

<table>
<thead>
<tr>
<th>Instance</th>
<th>LS-ECNF</th>
<th>Loandra</th>
<th>Open-WBO</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>cmin</td>
<td>cmin</td>
<td>cmin</td>
</tr>
<tr>
<td>30*12</td>
<td>103</td>
<td>360</td>
<td>191</td>
</tr>
<tr>
<td>30*16</td>
<td>131</td>
<td>N/A</td>
<td>806</td>
</tr>
<tr>
<td>30*20</td>
<td>182</td>
<td>N/A</td>
<td>965</td>
</tr>
<tr>
<td>30*24</td>
<td>195</td>
<td>2143</td>
<td>918</td>
</tr>
<tr>
<td>40*12</td>
<td>67</td>
<td>755</td>
<td>293</td>
</tr>
<tr>
<td>40*16</td>
<td>108</td>
<td>1185</td>
<td>382</td>
</tr>
<tr>
<td>40*20</td>
<td>126</td>
<td>2223</td>
<td>1084</td>
</tr>
<tr>
<td>40*24</td>
<td>148</td>
<td>4059</td>
<td>1279</td>
</tr>
<tr>
<td>50*12</td>
<td>64</td>
<td>2213</td>
<td>854</td>
</tr>
<tr>
<td>50*16</td>
<td>144</td>
<td>3424</td>
<td>1236</td>
</tr>
<tr>
<td>50*20</td>
<td>122</td>
<td>N/A</td>
<td>1426</td>
</tr>
<tr>
<td>50*24</td>
<td>225</td>
<td>N/A</td>
<td>2098</td>
</tr>
<tr>
<td>60*12</td>
<td>185</td>
<td>2740</td>
<td>1052</td>
</tr>
<tr>
<td>60*16</td>
<td>86</td>
<td>5202</td>
<td>1599</td>
</tr>
<tr>
<td>60*20</td>
<td>326</td>
<td>N/A</td>
<td>2137</td>
</tr>
<tr>
<td>60*24</td>
<td>131</td>
<td>N/A</td>
<td>2740</td>
</tr>
<tr>
<td>80*12</td>
<td>188</td>
<td>4472</td>
<td>1330</td>
</tr>
<tr>
<td>80*16</td>
<td>222</td>
<td>5003</td>
<td>2097</td>
</tr>
<tr>
<td>80*20</td>
<td>324</td>
<td>N/A</td>
<td>1599</td>
</tr>
<tr>
<td>80*24</td>
<td>319</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>100*16</td>
<td>108</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>100*20</td>
<td>126</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>100*24</td>
<td>148</td>
<td>4059</td>
<td>1279</td>
</tr>
<tr>
<td>120*12</td>
<td>64</td>
<td>2213</td>
<td>854</td>
</tr>
<tr>
<td>120*16</td>
<td>144</td>
<td>3424</td>
<td>1236</td>
</tr>
<tr>
<td>120*20</td>
<td>122</td>
<td>N/A</td>
<td>1426</td>
</tr>
<tr>
<td>120*24</td>
<td>225</td>
<td>N/A</td>
<td>2098</td>
</tr>
</tbody>
</table>

Table 2: Experimental results of comparing LS-ECNF with the state of the art solvers on nurse-rostering problem. Time limit is 3600s

<table>
<thead>
<tr>
<th>Instance</th>
<th>LS-ECNF</th>
<th>Loandra</th>
<th>Open-WBO</th>
<th>Gurobi</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>cmin</td>
<td>cmin</td>
<td>cmin</td>
<td>cmin</td>
</tr>
<tr>
<td>30*12</td>
<td>100</td>
<td>144</td>
<td>121</td>
<td></td>
</tr>
<tr>
<td>30*16</td>
<td>125</td>
<td>174</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>30*20</td>
<td>168</td>
<td>245</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>30*24</td>
<td>189</td>
<td>274</td>
<td>308</td>
<td></td>
</tr>
<tr>
<td>40*12</td>
<td>58</td>
<td>136</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>40*16</td>
<td>92</td>
<td>208</td>
<td>179</td>
<td></td>
</tr>
<tr>
<td>40*20</td>
<td>121</td>
<td>293</td>
<td>1483</td>
<td></td>
</tr>
<tr>
<td>40*24</td>
<td>142</td>
<td>343</td>
<td>385</td>
<td></td>
</tr>
<tr>
<td>50*12</td>
<td>59</td>
<td>139</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>50*16</td>
<td>140</td>
<td>433</td>
<td>296</td>
<td></td>
</tr>
<tr>
<td>50*20</td>
<td>114</td>
<td>1021</td>
<td>632</td>
<td></td>
</tr>
<tr>
<td>50*24</td>
<td>210</td>
<td>2154</td>
<td>1463</td>
<td></td>
</tr>
<tr>
<td>60*12</td>
<td>172</td>
<td>419</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>60*16</td>
<td>82</td>
<td>490</td>
<td>285</td>
<td></td>
</tr>
<tr>
<td>60*20</td>
<td>313</td>
<td>3506</td>
<td>1068</td>
<td></td>
</tr>
<tr>
<td>60*24</td>
<td>123</td>
<td>3796</td>
<td>1894</td>
<td></td>
</tr>
<tr>
<td>80*12</td>
<td>183</td>
<td>396</td>
<td>275</td>
<td></td>
</tr>
<tr>
<td>80*16</td>
<td>214</td>
<td>898</td>
<td>1884</td>
<td></td>
</tr>
<tr>
<td>80*20</td>
<td>308</td>
<td>2296</td>
<td>1680</td>
<td></td>
</tr>
<tr>
<td>80*24</td>
<td>319</td>
<td>4662</td>
<td>2238</td>
<td></td>
</tr>
<tr>
<td>100*16</td>
<td>72</td>
<td>2913</td>
<td>1476</td>
<td></td>
</tr>
<tr>
<td>100*20</td>
<td>130</td>
<td>7949</td>
<td>2662</td>
<td></td>
</tr>
<tr>
<td>100*24</td>
<td>105</td>
<td>9862</td>
<td>3444</td>
<td></td>
</tr>
<tr>
<td>120*12</td>
<td>433</td>
<td>4662</td>
<td>2238</td>
<td></td>
</tr>
<tr>
<td>120*16</td>
<td>121</td>
<td>7949</td>
<td>2662</td>
<td></td>
</tr>
<tr>
<td>120*20</td>
<td>123</td>
<td>3796</td>
<td>1894</td>
<td></td>
</tr>
<tr>
<td>120*24</td>
<td>185</td>
<td>N/A</td>
<td>7803</td>
<td></td>
</tr>
</tbody>
</table>

- TT-Open-WBO-inc: [Nadel, 2019] won two weighted categories of incomplete track in MSE 2019. We compare with the best version Open-WBO-Inc-BMO, and we denote it as Open-WBO for convenience. In addition, Open-WBO can also be used to solve Pseudo-Boolean problems. We mark this version as pb.
- CADICAL: [SAT Race 2019] ranked first in the SAT (satisfiable instances) track and second in the SAT+UNSAT track (of the SAT Race 2019).
- Gurobi: [Gurobi Optimization, 2019] is one of the most powerful mathematical optimization solvers. We use both the complete and heuristic versions which are denoted as comp and heur respectively.

**LS-ECNF** is implemented in C++ and compiled by g++ with -O3 option. Our experiments were conducted on a server using Intel Xeon E7-8850 v2 @2.30GHz, 2048GB RAM, running Ubuntu 16.04.5 Linux operation system. We adopt various time limits, including 300 seconds and 3600 seconds on nurse rostering problem, and the time limit is 3600 seconds on discrete tomography problems. For nurse rostering problem, in each run, the solver prints successively the best solution it has found so far. For each instance, **cmin** is the cost of the best feasible solution found. Discrete tomography problems are decision problems, so we show the time of finding the feasible solution of each instance. For each SAT and MaxSAT solver, we use three cardinality encodings marked as **car**, **seq**, **pig** respectively. In bold we present the best results for each instance.

### 5.1 Experiment Results on NRP

The experiment results on nurse rostering benchmarks are presented in Tables 1 and 2. When the time limit is 300 seconds, Gurobi (both complete and heuristic versions) fail to find feasible solutions on all these instances, so we do not report their results. **LS-ECNF** is the best solver, finding better solutions than other solvers, and usually much faster. When the time limit is 3600 seconds (Table 2), MaxSAT and Pseudo-Boolean algorithms still perform much worse than **LS-ECNF**. In the meantime, Gurobi can find a good feasible solution for the small instances. However, as the scale of the problem increases, the probability to find a feasible solution drops significantly. When the number of nurses increases to 100, Gurobi can hardly find any feasible solution.

Note that, Cadical is a SAT solver, so we do not report its results on this optimization problem. Also, SATLike fails to find feasible solutions on almost all instances, so its results are not reported, either.

### 5.2 Experiment Results on DTP

Experiment results on discrete tomography problem are presented in Table 3. The MaxSAT version of Open-WBO is
### 5.3 Experiment Analysis

In our experiments, GUP-based algorithm can improve the performance over *LS-ECNF* in more than 70% instances on NRP. And for DTP, using GUP-based algorithm can save more than 20% running time. Generally, techniques from SAT and MaxSAT can be beneficial for ESAT and EPMS solvers. Indeed, we can develop a DPLL framework for ECNF problems. During the propagation, a variable $x$ is picked to be assigned. For a clause $c$ s.t. $l \in c$ where $l$ is $x$ or its negation, if $l$ is true, then $\text{num_true}_c[c]$ is increased by 1, otherwise, $l$ is removed from $c$. If $|c| = \text{cardinality}[c]$, then $c$ is a generalized unit clause and all the rest literals in $c$ should be set to true. If $|c| < \text{cardinality}[c]$, which means there is a conflict, then backtracking is executed.

### 5.4 Relationship Between ECNF with CNF and PB

ECNF extends CNF by introducing the concept of cardinality. One can encode cardinality constraints into ECNF without any auxiliary variables and clauses. In Table 4, we present the average number of variables and clauses of each encoding. We can see that, encoding cardinality constraints into ECNF is more compact and efficient. Because that ECNF and CNF share almost the same data structure, techniques used for problems of CNF (e.g. SAT) can be used for problems of ECNF (e.g. ESAT). ECNF can be considered as a special case of Pseudo-Boolean Optimisation (PBO).

### 6 Conclusion

We proposed an extension of CNF formulas, which can easily deal with cardinality constraints without any auxiliary variable and clause. We also designed an efficient local search algorithm and a generalized UP-based algorithm for ECNF. Experiments results showed that our solver is much better than all state of the art SAT, MaxSAT, PB and ILP solvers for nurse rostering problems and discrete tomography problems. The strong results show that we opened a promising direction to solve cardinality constraints. We would like to encode other combinatorial problems with cardinality constraints into ECNF and design a more efficient solver.

### Acknowledgements

This work is supported by Beijing Academy of Artificial Intelligence (BAAI) and Youth Innovation Promotion Association, Chinese Academy of Sciences [No. 2017150].
References


