Towards Cardinality-Aware Local Search for SAT with Cardinality Constraints

Shuli Hu , **Dian Ling** , **Jiaqi Li** and **Minghao Yin***
School of Information Science and Technology, Northeast Normal University, China
{husl903, lingd246, lijq515, ymh}@nenu.edu.cn

Abstract

Satisfiability (SAT) with cardinality constraints arises naturally in many practical applications, where high-level counting requirements coexist with standard Conjunctive Normal Form (CNF) clauses. Translating these constraints into CNF can destroy structural information, limiting the effectiveness of search-based heuristics. In this paper, we propose a cardinality-aware local search framework to solve this problem, denoted as CardSAT-LS. CardSAT-LS integrates a preprocessing phase based on the generalized unit propagation and resolution, a cardinality-sensitive scoring function combining the make-break mechanism and cardinality violation, and an initialization based on fake-backbone variables. Furthermore, CardSAT-LS employs a unified framework that adaptively alternates between flip and swap operators when the search gets trapped in local optima. Finally, we conduct experiments on five public benchmarks from real-world applications as well as the MaxSAT and SAT competitions. Compared with ten state-of-the-art competitors, including SAT, MaxSAT, and PB solvers, CardSAT-LS solves the most instances with the lowest PAR-2 score. Additionally, we integrate CardSAT-LS into exact solvers for phase selection, which leads to significant speedups.

1 Introduction

Satisfiability problem (SAT) is a basic and important constraint problem, and it constitutes a central paradigm for modeling and solving a wide range of real-world problems, including planning [Rintanen *et al.*, 2006], scheduling [Horbach, 2010], and verification [Vizel *et al.*, 2015]. Over the past decades, remarkable success has been achieved in solving techniques for SAT, including the complete solvers based on conflict-driven clause learning [Mull *et al.*, 2022; Chowdhury *et al.*, 2020; Biere and Fröhlich, 2015] and incomplete methods such as stochastic local search [Hutter *et al.*, 2002; Bailleux and Boufkhad, 2003; Cai *et al.*, 2015]. These techniques played a crucial role in enabling SAT-based approaches to tackle large and complex real-world instances.

Although many aspects of the SAT solvers have changed, most modern solvers still use Conjunctive Normal Form (CNF) formulas as their input representation. However, many practical problems involve constraints that go beyond simple logical disjunctions [Bard *et al.*, 2007; Abío and Stuckey, 2014]. For such problems, it is challenging for the users to find an appropriate encoding. Moreover, essential structural information captured by high-level constraints is hard to express at the clause level. Among these high-level constraints, cardinality constraints are particularly common in SAT problems, which assert the sum of a set of literals exceeds a given bound [Marques-Silva and Lynce, 2007]. Cardinality constraints arise naturally in any problems that require counting, such as those with resource or assignment limitations.

Despite the richer representations provided by Satisfiability Modulo Theories (SMT) and Pseudo-Boolean (PB) constraints, which offer more expressive modeling and stronger reasoning, eager approaches that translate constraints into SAT are still desirable in many practical settings. The reason is that in some domains higher-level reasoning is not always necessary, and eager solving can be surprisingly effective. For example, for bit-vectors the eager SMT solver performs well in practice [Niemetz and Preiner, 2023]. Besides, SAT solvers benefit from decades of intensive optimization, including efficient propagation, conflict-driven clause learning, and sophisticated heuristics, which make them highly competitive on large-scale instances. In this context, SAT with cardinality constraints, allowing cardinality constraints to be represented explicitly, bridges the gap between the pure CNF and fully general PB or SMT.

Many approaches for handling cardinality constraints have been proposed. A common approach is to encode cardinality constraints into CNF using dedicated coding, such as sequential counters[Sinz, 2005], cardinality networks[Asín *et al.*, 2009; Asín *et al.*, 2011], and totalizer[Bailleux and Boufkhad, 2003; Ogawa *et al.*, 2013; Ignatiev *et al.*, 2018]. While this enables the use of highly optimized SAT solvers, a well-known drawback is that many auxiliary variables and clauses are introduced, which can obscure the constraint's original structure [Wynn, 2018]. Interestingly, recent work by [Reeves *et al.*, 2025] demonstrates that for such encodings, the ordering of literals often has a greater impact on

*Corresponding author.

solver performance than the choice of the encoding method itself. Another approach is using PB solvers, which handle cardinality constraints directly, including Stochastic Local Search (SLS) PB solvers [Lei *et al.*, 2020; Lei *et al.*, 2021; Chu *et al.*, 2023] and exact PB solvers [Devriendt *et al.*, 2021; Le Berre and Parrain, 2011]. However, a significant practical limitation is that PB solvers perform poorly when the input is provided in standard CNF format[Elffers and others, 2020]. Recently, [Reeves *et al.*, 2024] has proposed an extension of CDCL with native propagation on cardinality constraints, which performed well on SAT formulas. Despite these advances, existing local search methods are not well-tailored to instances combining CNF clauses and cardinality constraints. This motivates our cardinality-aware local search approach.

Contributions. We present CardSAT-LS, a cardinality-aware local search solver for SAT with At-Least-K CNF constraints. It combines generalized preprocessing, a cardinality-aware scoring function, fake-backbone-based initialization, and adaptive flip/swap moves. Experiments on 814 benchmark instances show that it solves the most instances and achieves the best PAR-2 among 10 competing SAT, MaxSAT, and PB solvers, and it also accelerates exact solvers when used for phase selection.

2 Preliminaries

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of Boolean variables. A literal is either a variable x_i (positive literal) or its negation $\neg x_i$ (negative literal). The *phase* of a literal indicates whether it is positive or negative. Let $L = \{l_1, l_2, \ldots, l_m\}$, and $\texttt{var}(l)$ is to denote the corresponding variable of a literal. A *clause* is a disjunction of literals, and a *Conjunctive Normal Form* (CNF) F is a conjunction of clauses. A *pure literal* is a literal whose complement (the same variable with the opposite sign) does not appear anywhere in the formula.

An *assignment* α is a mapping from variables to truth values in $\{0, 1\}$, where 1 denotes *true* and 0 denotes *false*. An assignment α satisfies a positive literal l by setting $\alpha(\texttt{var}(l))$ to 1, and satisfies a negative literal l by setting $\alpha(\texttt{var}(l))$ to 0. An assignment falsifies a literal if the corresponding condition does not hold. An assignment α satisfies a clause if it contains at least one literal satisfied by α. A CNF F is *satisfiable* if and only if there exists an assignment that satisfies all clauses, and unsatisfiable otherwise.

A *unit* is a clause containing a single literal. *Unit Propagation* (UP) applies the following operation until a fixed-point: Given a CNF formula F, all unit clauses are collected and the assignment α is derived by setting each variable to satisfy its unit clause. That is, if the unit $\{x_i\}$ appears in F, we set x_i to true; and if the unit $\{\neg x_i\}$ appears in F, we set x_i to false. All clauses satisfied by α are removed from F, and all literals falsified by α are removed from the remaining clauses. A conflict is detected if UP yields the empty clause ($\bot$).

2.1 Cardinality Constraints

A *cardinality constraint* over Boolean variables has the form $\sum_{i=1}^{r} x_i \triangleright k$, where $x_i \in \{0, 1\}, \triangleright \in \{\geq, \leq, =\}$, and $k \in \mathbb{N}$. Such a constraint is satisfied by an assignment if the sum of the values of the variables satisfies the relation $\triangleright k$.

Since each literal l can be identified with its underlying variable $\texttt{var}(l)$, we equivalently write cardinality constraints over literals in the form $\sum_{i=1}^{r} l_i \triangleright k$, where each l_i is a literal interpreted as a Boolean value in $\{0, 1\}$.

For clarity and without loss of generality, constraints of the form $\sum_{i=1}^{r} l_i \leq k$ can be rewritten as $\sum_{i=1}^{r} \neg l_i \geq r - k$, since $l_i + \neg l_i = 1$ holds. Similarly, constraints of the form $\sum_{i=1}^{r} l_i = k$ can be rewritten as $(\sum_{i=1}^{r} l_i \geq k) \wedge (\sum_{i=1}^{r} \neg l_i \geq r - k)$. Therefore, in the following we restrict our attention to *At-Least-K* (ALK) constraints of the form $\sum_{i=1}^{r} l_i \geq k$.

2.2 At-Least-K Conjunctive Normal Form (KNF)

For the SAT with cardinality constraints, we adopt the same formulation of [Reeves *et al.*, 2024]. Following its terminology, we refer to ALK constraints ($\sum_{i=1}^{r} l_i \geq k$) as *klauses*, each consisting of a set of literals and a bound k. A standard clause can be viewed as a special case of a klause with bound $k = 1$, that is, the *At-Least-One* (ALO) constraint. A formula is in *At-Least-K Conjunctive Normal Form* (KNF) if it is a conjunction of klauses. Under the KNF representation, cardinality constraints are explicitly represented, enabling reasoning directly at the level of constraints rather than at the level of clauses. Note that CNF is a strict subset of KNF.

3 Cardinality-Aware Local Search for KNF

In this section, we introduce our proposed cardinality-aware local search for SAT with cardinality constraints represented in KNF. Firstly, the generalized UP and resolution are provided to simplify the formula in the preprocessing phase. Secondly, a novel score function combining cardinality violation and the make-break mechanism is introduced. Subsequently, an initialization based on fake-backbone variables is presented. Finally, we propose a unified framework that adaptively alternates between flip and swap operators.

3.1 Generalized UP and Resolution

Definition 1 (Generalized Unit)**.** *Given a klause $c := \sum_{i=1}^{r} l_i \geq k$, if $k = |c|$, where $|c|$ is the number of literals in c, we say that c is a generalized unit.*

Unit Propagation can be generalized to KNF via Definition 1, and the generalized UP is provided in Theorem 1.

Theorem 1 (Generalized UP)**.** *Given a generalized unit klause $c := \sum_{i=1}^{r} l_i \geq k$, it holds if l_i is true for all $1 \leq i \leq r$.*

Proof. The proof is straightforward: when $k = |c|$, satisfying klause c requires all of its literals to be true. □

The process of Generalized UP is similar to original UP. After that, some variables' assignments could be fixed, and some klauses and literals can be removed to simplify the formula.

Theorem 2 (Generalized Resolution)**.** *Given two klauses $c_1 := y + \sum_{i=1}^{r} x_i \geq k_1$ and $c_2 := \neg y + \sum_{j=1}^{t} z_j \geq k_2$, if the sets of literals $\{x_i\}_{i=1}^{r}$ and $\{z_j\}_{j=1}^{t}$ are disjoint*

($\{x_i\}_{i=1}^r \cap \{z_j\}_{j=1}^t = \emptyset$), then their resolvent c with respect to y is given by:

$$c \coloneqq c_1 \otimes_y c_2 = \sum_{i=1}^{r} x_i + \sum_{j=1}^{t} z_j \geq k_1 + k_2 - 1 \quad (1)$$

where c remains a valid klause.

Proof. From $c_1 \wedge c_2$, adding the two inequalities yields:

$$\begin{aligned} (y + \sum_{i}^{r} x_i) + (\neg y + \sum_{j}^{t} z_j) &\geq k_1 + k_2 \\ (y + \neg y) + (\sum_{i}^{r} x_i + \sum_{j}^{t} z_j) &\geq k_1 + k_2 \\ \sum_{i}^{r} x_i + \sum_{j}^{t} z_j &\geq k_1 + k_2 - 1 \quad (2) \end{aligned}$$

Equation (2) holds due to the tautology $y + \neg y = 1$. Since $\{x_i\}_{i=1}^r \cap \{z_j\}_{j=1}^t = \emptyset$, the summation $\sum_{i=1}^r x_i + \sum_{j=1}^t z_j$ contains no duplicated literals. Consequently, the resolvent $c \coloneqq c_1 \otimes_y c_2 = \sum_{i=1}^r x_i + \sum_{j=1}^t z_j \geq k_1 + k_2 - 1$ is a well-formed klause, confirming that c is a logical consequence of the resolution of c_1 and c_2. □

We generalized the Resolution Checking (RC) [Chen *et al.*, 2022], a variant of the Bounded Variable Elimination (BVE) [Biere, 2013] with stricter conditions, to the KNF setting. To address the time consumption of BVE, we employ a simple heuristic to decide whether to perform the elimination. Assume $s(l)$ denotes the set of klauses in which literal l appears. For each variable x, if $|s(x)| \times |s(\neg x)| \leq |s(x)| + |s(\neg x)|$, we resolve $s(x)$ with $s(\neg x)$ and check whether all resulting klauses are tautologies. If so, the clauses related to variable x can be removed.

During elimination, many *pure literals* could be produced, so we propose a generalized pure literal rule to simplify the formula again. The process is illustrated in Example 1.

Example 1. *Given a KNF $F = (\neg x_3 + \neg x_5 + \neg x_7 + \neg x_8 \geq 2) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_5 \vee \neg x_7)$, we could easily eliminate variable x_1 since $(x_1 \vee x_2 \vee x_3) \otimes_{x_1} (\neg x_1 \vee \neg x_2 \vee x_3)$ is always true. Thus, the simplified KNF $F' = (\neg x_3 + \neg x_5 + \neg x_7 + \neg x_8 \geq 2) \wedge (x_5 \vee \neg x_7)$.*

Furthermore, we observe that $\neg x_3$ is a pure literal (since x_3 does not appear positively). According to the pure literal rule, we can safely set $\neg x_3$ to true.

Substituting this assignment into the cardinality constraint $(\neg x_3 + \neg x_5 + \neg x_7 + \neg x_8 \geq 2)$ simplifies it to $(\neg x_5 + \neg x_7 + \neg x_8 \geq 1)$. Thus, F' is further simplified to: $(\neg x_5 + \neg x_7 + \neg x_8 \geq 1) \wedge (x_5 \vee \neg x_7)$

The preprocessing algorithm based on the generalized UP and resolution is outlined in Algorithm 1, and described below. The algorithm first performs a generalized unit propagation (lines 2-9) based on Theorem 1; unlike the traditional UP, we decrement the bound k of a klause when a literal in it is assigned true and remove the klause if $k = 0$. After Generalized-UP, we apply the Generalized Resolution Check and Generalized Pure Literal Rule (line 10-23) as shown in Example 1. Note that the resolution is applied exclusively during preprocessing to simplify the formula while preserving its KNF structure. The resulting formula is then solved by CardSAT-LS local search procedure.

Algorithm 1 Generalized-Preprocessing

Require: A KNF F
Ensure: A simplified KNF F'
1: $F' \leftarrow F$
2: **while** $\exists$ generalized unit c **do**
3: $\quad \forall l_i \in c, l_i \leftarrow 1$
4: $\quad \forall c_j \in s(\neg l_i), c_j \leftarrow c_j \setminus \{\neg l_i\}$
5: $\quad \forall c_j \in s(l_i), c_j, k_j \leftarrow c_j \setminus \{l_i\}, k_j - 1$
6: $\quad$ **if** $k_j = 0$ **then**
7: $\quad\quad \forall c_j \in s(l_i), F' \leftarrow F' \setminus \{c_j\}$
8: $\quad$ **end if**
9: **end while**
10: **for** each variable $x \in F'$ **do**
11: $\quad$ **if** x is the pure literal and $x \in c_j$ (resp. $\neg x$) **then**
12: $\quad\quad x \leftarrow 1$
13: $\quad\quad c_j, k_j \leftarrow c_j \setminus \{x\}, k_j - 1$
14: $\quad\quad$ **if** $k_j \leq 0$ **then**
15: $\quad\quad\quad F' \leftarrow F' \setminus \{c_j\}$
16: $\quad\quad$ **end if**
17: $\quad$ **else if** $|s(x)| \times |s(\neg x)| \leq |s(x)| + |s(\neg x)|$ **then**
18: $\quad\quad \forall c_i \in s(x), \forall c_j \in s(\neg x), R \leftarrow R \cup c_i \otimes_x c_j$
19: $\quad\quad$ **if** $\forall r \in R$ is tautology **then**
20: $\quad\quad\quad F' \leftarrow F' \setminus \{c \in s(x) \cup s(\neg x) \mid k(c) = 1\}$
21: $\quad\quad$ **end if**
22: $\quad$ **end if**
23: **end for**
24: **return** F'

3.2 Scoring Function Combining Make-break Mechanism and Violation

Scoring Function in SLS SAT Solver. Usually, SLS algorithms for SAT select a variable to flip according to *scoring functions* [Selman *et al.*, 1994] based on the properties $make(x)$ and $break(x)$, which are the number of clauses that would become satisfied and unsatisfied by flipping a variable x. In this context, $score(x) = \sum_{c \in Make(x)} w(c) - \sum_{c \in Break(x)} w(c)$, where $Make(x)$ and $Break(x)$ are the sets of clauses impacted by flipping x, and $w(c)$ is the weight of clause c [Morris, 1993]. However, this scoring function has a drawback: it cannot measure the *slack* of cardinality constraints. In the following, we illustrate our intuition through a simple SAT instance with cardinality constraints.

Example 2. *Let us consider a KNF instance, which consists of four clauses and one cardinality constraint: $\mathcal{F} = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_4) \wedge (x_2 \vee x_5) \wedge (x_3 \vee x_6) \wedge (x_1 + x_3 + x_4 + x_5 + x_6 \geq 3)$, the current assignment is $\alpha = (0, 0, 0, 1, 0, 0)$ and all weights of klauses are 1. Calculating the scores, we find that $score(x_2) = 2$ and $score(x_3) = 2$.*

Example 2 shows a limitation: although flipping x_3 is superior to flipping x_2 for satisfying the cardinality constraint,

Algorithm 2 Initialization

Ensure: An assignment α
1: $\alpha \leftarrow \emptyset$
2: **for** each variable $x_i \in F$ **do**
3: **if** time_stamp$[x_i] > thres_{bb}$ **then**
4: $\alpha \leftarrow \alpha \cup \{x_i \mapsto 0\}$
5: **else**
6: $\alpha \leftarrow \alpha \cup \{x_i \mapsto$ backbone$[x_i]\}$
7: **end if**
8: **end for**
9: **return** α

the algorithm cannot distinguish them and picks one randomly.

To address this issue, we introduce a metric to quantify the satisfaction level of cardinality constraints. Specifically, we adapt the concept of violation degree from SLS PBO solvers [Lei *et al.*, 2021] as a scoring component. Integrating this metric with the traditional make-break mechanism, we construct a novel scoring function. It consists of two parts: (1) $score_{m\_b}$ is to evaluate the impact on the normal clause by flipping x; (2) $score_{card}$ is to evaluate the decrement of the violation of cardinality constraints.

$$score(x) = score_{m\_b}(x) + score_{card}(x) \tag{3}$$

$$\begin{aligned} &= \sum_{c \in Make(x)} w(c) - \sum_{c \in Break(x)} w(c) \\ &+ \sum_{c \in card(x)} w(c) \cdot |\Delta viol(c)| \end{aligned} \tag{4}$$

where $viol(c) = \max(0, k - \sum_i^r l_i)$. Revisiting Example 2, based on the new scoring function, $score(x_2) = 2$ and $score(x_3) = 3$.

Like state-of-the-art solvers, we employ dynamic weighting at the klauses level to escape local optima. The rules are as followings.

- **Initialization_rule**: at the start of the search process, the weight of each klause c is initialized as 1, i.e., $w(c) \coloneqq 1$.
- **Update_rule**: when the search is trapped in a local optimum (there is no variable whose $score$ value is greater than 0), for each falsified klause c, $w(c) \coloneqq w(c) + 1$.

3.3 Initialization Based on Fake-Backbone Variables

In traditional SLS SAT solvers, initialization is often overlooked, with randomized assignments typically yielding satisfactory performance. However, in SLS PBO solvers, this paradigm shifts toward an all-zero initialization [Lei *et al.*, 2021]. Nevertheless, we observe that preprocessing significantly biases the phase selection of variables. To balance diversity with the exploitation of historical search information during restarts, we draw inspiration from the *target* phase strategy in CaDiCaL [Biere, 2017] (which tracks the longest conflict-free assignment sequence). Consequently, we propose a *fake-backbone* randomized initialization strategy to substantially enhance solver performance.

Algorithm 3 The CardSAT-LS Algorithm

Require: A KNF F
Ensure: "SAT" and its model, or "Unknown"
1: $F \leftarrow$ `Generalized-Preprocessing`(F)
2: **while** no terminating criteria are met **do**
3: $\alpha \leftarrow$ `Initialization`()
4: $\forall c \in F,\ w(c) \coloneqq 1$
5: non_improved $\leftarrow 0$
6: **for** $step \leftarrow 0; step < L; step++$ **do**
7: **if** $\nexists$ unsatisfied klauses **then**
8: **return** {"SAT", α}
9: **end if**
10: **if** the number of satisfied klauses increases **then**
11: non_improved $\leftarrow 0$
12: **end if**
13: non_improved $\leftarrow$ non_improved + 1
14: **if** $\exists$satisfied cardinality constraints $\wedge$ non_improved $> thres_{n\_i} \wedge p < p_{swap}$ **then**
15: `Swap-Operator`($step$,α)
16: **else**
17: **if** $D \coloneqq \{x | score(x) > 0 \wedge cc[x] > 0\} \neq \emptyset$ **then**
18: $v \leftarrow$ a variable in D with the highest $score$
19: **else**
20: update klauses weights by the weighting rules
21: **if** $\exists$ unsatisfied klauses **then**
22: $c \leftarrow$ a random unsatisfied klause
23: $v \leftarrow$ the variable whose literal is false in c with the highest $score$
24: **end if**
25: **end if**
26: `flip`(v), time_stamp$[v] = step$
27: **end if**
28: **end for**
29: backbone $\leftarrow \alpha$
30: **end while**
31: **return** "Unknown"

Given that SLS algorithms lack a deterministic mechanism to identify actual backbones, our initialization strategy is predicated on a simple yet effective heuristic assumption: if a variable's last flip occurred at a very early stage of the search (i.e., its timestamp is small), it suggests that the variable's current assignment is likely correct and should remain unaltered. Thus, we retain the assignments of these variables to exploit historical search knowledge while restarting. For the first initialization, we employ a traditional randomized approach to ensure a robust and unbiased starting point for the solver. The initialization is as detailed in Algorithm 2.

3.4 The CardSAT-LS Algorithm

We develop a new SLS algorithm named CardSAT-LS, which is based on the main ideas proposed in previous sections. CardSAT-LS adopts the iterated local search with Configuration Check [Cai *et al.*, 2015] to avoid revisiting the same variable frequently. The pseudo-code is outlined in Algorithm 3.

CardSAT-LS starts by performing generalized preprocessing to simplify the formula (Line 1). It then iteratively executes the local search procedure, adaptively alternating be-

Algorithm 4 Swap-Operator

Require: Current step $step$, assignment α
1: $c \leftarrow$ select a satisfied cardinality constraint randomly
2: $D := \{x | score(x) > 0 \wedge cc[x] > 0 \wedge (x \in c \vee \neg x \in c)\}$
3: $v_{in} \leftarrow$ a variable in D with the highest $score$ and $\alpha(x)$ is 1;
4: $v_{out} \leftarrow$ a variable in D with the highest $score$ and$\alpha(x)$ is 0;
5: `flip`(v_{in}) and `flip`(v_{out})
6: time_stamp[v_{in}] = $step$, time_stamp[v_{out}] = $step$

tween *Flip* and *Swap* operators until a stopping criterion is met. In the local search process, an initial assignment is generated by the method described in Section 3.3 (Line 3). CardSAT-LS then initializes the weights of all klauses to 1. The algorithm then enters the main search procedure (Lines 6-28).

In each search step, our algorithm selects and flips variables based on the following logic: (I) If there exists satisfied cardinality constraints but the search has stagnated, the algorithm invokes the `Swap Operator` described below with a probability p_{swap}. (II) If the swap condition is not met, the algorithm checks the variable set D that satisfy $score(x) > 0$ and configuration check ($cc[x] > 0$). If D is not empty, the variable with the highest $score$ is selected for flipping, breaking ties by preferring the variable that has been flipped least recently. (III) When D is empty, indicating that the search is trapped in a local optimum, the algorithm updates the weights of all klauses according to the dynamic weighting rules. After the update, if unsatisfied klauses remain, the algorithm randomly picks an unsatisfied klause c. To reduce the violation, it selects the variable in c whose literal is false under the current assignment and which possesses the highest $score$. Note that our algorithm employs the *BMS* (Best from Multiple Selections) strategy [Cai *et al.*, 2017] to select variables and klauses. After flipping or swapping, CardSAT-LS will update the $score$ and cc of those affected variables.

Finally, the search continues until a satisfying assignment α is found (returning "SAT") or the step limit L is reached.

A New Swap-Operator. Compared to standard clauses, cardinality constraints impose significantly more stringent restrictions on variables, which often leads the local search to become trapped in local optima. Specifically, once a cardinality constraint is exactly satisfied, the solver encounters an issue: it becomes reluctant to flip variables that would violate the constraint, even if such flips could potentially satisfy a greater number of clauses. To mitigate this, we introduce a *swap* operator. This operator is specifically triggered when the search stagnates in a local optimum while there exists a satisfied cardinality constraint, as detailed in Lines 14-15 of Algorithm 3.

4 Experimental Evaluation

In this section, we introduce the experimental preliminaries and our results. We compare CardSAT-LS with 10 state-of-the-art solvers to verify its effectiveness. Then, we conduct the experiment to analyze the effectiveness of main ideas. Finally, we integrate CardSAT-LS into complete solvers for phase selection, which leads to significant speedups.

Benchmarks. We evaluate our algorithm on 5 benchmarks, including 2 competition benchmarks and 3 real-world application benchmarks.

- **MaxSAT24:** The benchmark is from the latest 2024 MaxSAT competition unweighted track [Berg *et al.*, 2024]. We generate 368 satisfiable SAT problems by making the cardinality constraint's bound the known optimum.
- **SAT25:** This benchmark is derived from the main track of the latest 2025 SAT Competition [Codel *et al.*, 2025]. We applied the cardinality extraction procedure [Reeves *et al.*, 2024] to each formula, and generated 273 instances (82 known UNSAT instances) with explicit cardinality constraints.
- **DES:** 120 instances are from diagnosis of Discrete-Event System problems [Anbulagan and Grastien, 2009] [1], with 14,621 to 636,593 variables and 3,320 to 142,474 constraints.
- **MVC:** 117 Minimum Vertex Cover instances are from the Second DIMACS Challenge and large real-world graphs[2], with 34 to 58,790,782 variables and 112 to 150,998,977 constraints.
- **WSNO:** 18 instances are from Wireless Sensor Network Optimization (WSNO) [Kovásznai *et al.*, 2018; Kovásznai *et al.*, 2019][3], encoded following [Lei *et al.*, 2021] into KNF, with 58,580 to 2,311,113 variables and 214,679 to 26,701,722 constraints.

State-of-the-Art Competitors. We compare our algorithm against 10 state-of-the-art algorithms including SAT, MaxSAT and Pseudo-Boolean solvers.

- CCDCL [Reeves *et al.*, 2024]: a CDCL-based SAT solver built on CaDiCaL [Biere, 2017] with native cardinality constraint propagation.
- CCAnr [Cai *et al.*, 2015]: a local search SAT solver with configuration checking.
- TaSSAT [Chowdhury *et al.*, 2024]: the best local search SAT solver that effectively solves hard combinatorial problems.
- YalSAT [Biere, 2017]: an efficient implementation of the ProbSAT local search algorithm.
- WalkSATlm [Cai *et al.*, 2013]: an improved version from the famous WalkSAT(SKC) algorithm.
- RoundingSAT [Devriendt *et al.*, 2021]: an exact PBO solver combining core-guided search with cutting planes reasoning.
- NuPBO [Chu *et al.*, 2023]: a local search PBO solver, incorporating the new scoring function and weighting scheme.

[1] http://grastien.net/ban/data/cc/
[2] https://lcs.ios.ac.cn/%7Ecaisw/graphs.html
[3] https://lcs.ios.ac.cn/%7Ecaisw/Resource/LS-PBO/benchmark/

- DLS-PBO [Chen *et al.*, 2024]: a local search PBO solver, combining a dynamic scoring function.
- NuPBO-DeepOPT+ [Zhao *et al.*, 2025]: the state-of-the-art local search solver for PBO.
- Open-WBO [Martins *et al.*, 2014]: a complete unsatisfiable-core-based MaxSAT solvers.

Experimental Setup. CardSAT-LS [4] is implemented in C++, and compiled with g++ (version 11.4.0) using the option "-O3 -flto". All the experiments were performed on a workstation under the operating system Ubuntu 22.04.5, with the Intel(R) Xeon(R) Platinum 8260 2.40GHz CPU and 1TB RAM. The cut-off time for all solvers is 3600 seconds. Each run was allocated a memory limit of 30 GB, following the SAT Competition rules.

For parameter tuning, we employed Sequential Model-based Algorithm Configuration (SMAC) [Lindauer *et al.*, 2022], conducting the tuning on 300 instances randomly selected from all the benchmarks, with a time limit set to 300 seconds. The parameter values obtained after tuning are as follows: the maximum step L is 10^9; the probability of calling swapping operator p_{swap} is 0.32; the threshold for fake-backbone $thres_{bb}$ is 5076; the threshold of not improved steps $thres_{n\_i}$ is 122. For reproducibility, we set all randomization seeds to 20.

4.1 Results on Competition Benchmarks

Table 1 presents the comparative performance of CardSAT-LS against 10 state-of-the-art competitors on five benchmarks, comprising 3 exact solvers and 7 SLS solvers. For each solver, we report the number of solved instances and PAR-2 score. PAR-2 is the sum of runtime over all instances, where timeouts and memory-outs are penalized by 2×3600 seconds, divided by the total number of instances.

On the MaxSAT24 benchmark, CardSAT-LS exhibits dominant performance, solving a total of 291 instances. This surpasses the best exact solver, CCDCL (258 solved), and the leading LS competitor, DLS-PBO (243 solved). In terms of solving efficiency, CardSAT-LS achieves the lowest PAR-2 score of 1605.58, 30.2% lower than the second-best score.

On SAT25, all three exact solvers solve more instances than the SLS solvers. Among SLS solvers, CardSAT-LS and CCAnr tie for second in solved count (46 each), behind TaSSAT (61). The corresponding PAR-2 scores are 5606.86, 5523.68, and 5030.42, respectively, indicating that SAT25 remains a comparatively difficult benchmark for CardSAT-LS.

Across all 814 instances, Figure 1 shows that the cumulative curve of CardSAT-LS stays above all competitors throughout the measured range. It solves 234, 348, 456, and 536 instances within 1, 10, 100, and 1000 seconds, versus the best competing counts of 190, 296, 399, and 481. At the cutoff, it solves 557 instances, 32 more than CCDCL, and its aggregate PAR-2 of 2369.06 is 12.5% lower than CCDCL's 2708.11.

[4] https://github.com/virgiling/CardSAT-LS

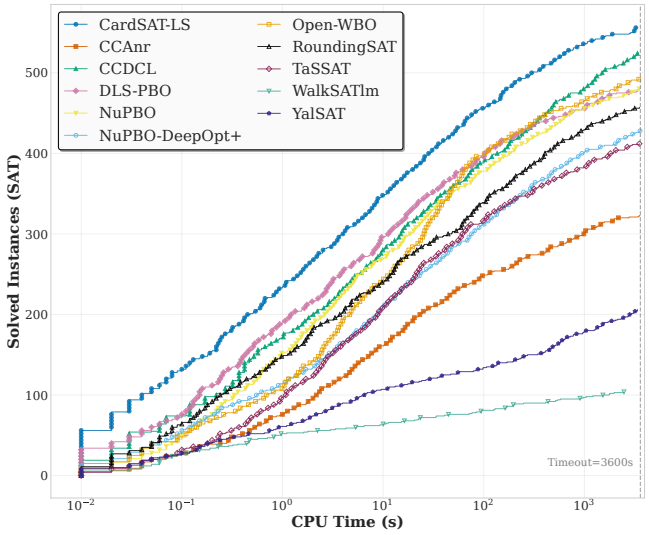

Figure 1: Cumulative runtime distributions of 11 solvers on the 814 instances in Table 1. The time axis is logarithmic, and the cutoff is 3600 seconds.

4.2 Results on Real-World Application Benchmarks

Table 1 shows that CardSAT-LS achieves or ties the highest solved count on all three application benchmarks. On DES, both CardSAT-LS and TaSSAT solve all 120 instances; their PAR-2 scores are 61.81 and 28.77, respectively, while TaSSAT solves no WSNO instance. On MVC, CardSAT-LS solves 82 instances, four more than TaSSAT and 31 more than the best exact solver; its PAR-2 of 2213.67 is 10.9% lower than the second-best score of NuPBO. On WSNO, CardSAT-LS solves all 18 instances and achieves the lowest PAR-2 of 13.02, followed by CCDCL at 15.19. These results demonstrate broad coverage across the three application domains without implying uniform dominance on every metric.

4.3 Analysis on the Underlying Ideas

Figure 2 compares five CardSAT-LS configurations on all 814 benchmark instances:

- CardSAT-LS+P+B+S: retains all three strategies.
- CardSAT-LS-P-B-S: disables P, B, and S and uses NuPBO's all-zero initialization.
- CardSAT-LS+P-B-S: retains only generalized preprocessing.
- CardSAT-LS-P+B-S: retains only fake-backbone initialization.
- CardSAT-LS-P-B+S: retains only the swap operator.

As shown in Figure 2, each single-strategy variant improves both metrics over the base variant (486 solved, PAR-2 2999.67). Preprocessing gives the largest individual gain, solving 549 instances and reducing PAR-2 to 2453.64, an increase of 63 solved instances and an 18.2% PAR-2 reduction. Fake-backbone initialization and the swap operator also improve the base variant, reaching 521/2682.85 and 527/2673.94, respectively. Thus, every strategy contributes

	MaxSAT24 (368)		**SAT25** (191)		**DES** (120)		**MVC** (117)		**WSNO** (18)		**Total** (814)	
	#Solved	*PAR-2*	*#Solved*	*PAR-2*	*#Solved*	*PAR-2*	*#Solved*	*PAR-2*	*#Solved*	*PAR-2*	*#Solved*	*PAR-2*
CCDCL	*258*†	*2298.72*†	***102***	***3651.54***	106	914.14	41	4709.90	***18***	*15.19*†	*525*†	*2708.11*†
Open-WBO	249	2392.19	70†	4823.27†	*116*	*309.85*	41	4678.93	17†	470.97	493	2941.85
RoundingSAT	214	3087.09	66	4928.87	108	787.86	*51*	*4154.86*	***18***	69.88	457	3267.05
CardSAT-LS	***291***	***1605.58***	46	5606.86	***120***	61.81†	***82***	***2213.67***	***18***	***13.02***	***557***	***2369.06***
CCAnr	117	5015.79	46	5523.68	109	711.05	39	4820.96	12	2673.19	323	4420.55
TaSSAT	153	4315.87	*61*	*5030.42*	***120***	***28.77***	78†	2505.93	0	7200.00	412	3655.15
WalkSATlm	42	6389.21	29	6178.23	0	7200.00	33	5176.52	0	7200.00	104	6302.86
YalSAT	86	5587.92	37	5900.25	35	5321.93	35	5063.91	13	2350.21	206	5475.08
DLS-PBO	*243*	*2523.41*	25	6304.54	*119*†	*180.14*	73	2754.49	***18***	*115.68*	478	*3045.16*
NuPBO	239	2637.27	*38*	*5855.19*	108	863.90	*77*	*2485.32*†	***18***	181.06	*480*	3054.75
NuPBO-DeepOpt+	206	3312.50	24	6361.84	106	1084.96	75	2603.00	***18***	122.59	429	3527.11

Table 1: Results of all solvers on all benchmarks. The first three are exact solvers. Best results are in bold, second-best are marked with †, and the best within each category is in italic.

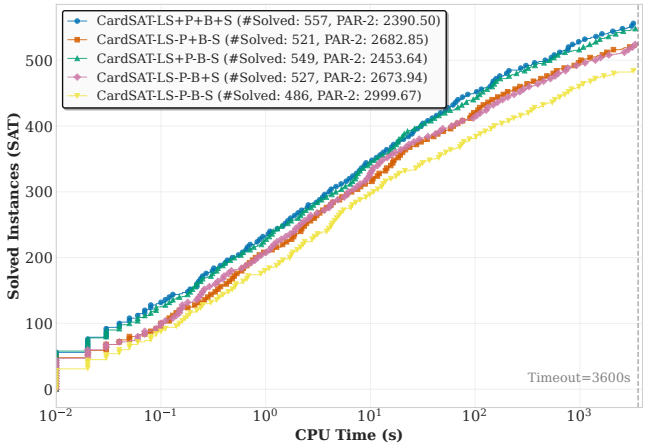

Figure 2: Runtime distributions of CardSAT-LS variants.

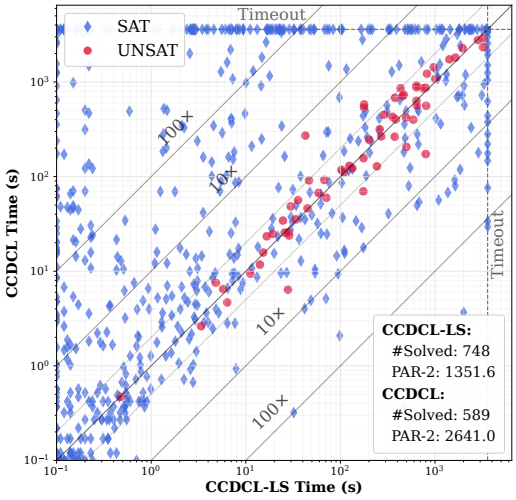

Figure 3: Per-instance runtimes of CCDCL-LS and CCDCL on all 896 instances.

individually, and their combination provides an additional empirical gain.

4.4 Integration with CCDCL

We integrated our proposed local search algorithm into the CCDCL framework (CCDCL+CardSAT-LS, CCDCL-LS in short), specifically employing it to guide the solution search and determine phase selection during each walk iteration. The comparative results are presented in Figure 3.

Across all 896 instances, CCDCL-LS solves 748 versus 589 instances and lowers PAR-2 from 2641.04 to 1351.61.

5 Conclusions and Future Work

This paper proposes a cardinality-aware local search framework for solving SAT with cardinality constraints. First, we introduced a generalized preprocessing technique along with an initialization method specifically designed to cooperate with it. Furthermore, we proposed a cardinality-sensitive scoring function and a novel swap operator which effectively guides the direction of search. Based on these ideas, we developed a new SLS solver named CardSAT-LS. Across 814 instances, CardSAT-LS achieves the highest solved count and lowest aggregate PAR-2 among the 11 evaluated solvers. Additionally, we integrate CardSAT-LS into exact solvers for phase selection, which leads to significant speedups. For future work, we would like to develop more efficient heuristic strategies and explore the effect of instance features on the performances of different categories of SAT with cardinality constraints.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant Nos. 62106040, 62532014), Jilin Science and Technology Association (Grant No. QT202320), and the Science and Technology Development Plan Project of Jilin Province of China (Grant No. 20240602108RC).

References

[Abío and Stuckey, 2014] Ignasi Abío and Peter J Stuckey. Encoding linear constraints into sat. In *International Conference on Principles and Practice of Constraint Programming*, pages 75–91. Springer, 2014.

[Anbulagan and Grastien, 2009] Anbulagan and A. Grastien. Importance of variables semantic in CNF encoding of cardinality constraints. In *Eighth Symposium on Abstraction, Reformulation and Approximation (SARA-09)*, 2009.

[Asín *et al.*, 2009] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks and their applications. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 167–180. Springer, 2009.

[Asín *et al.*, 2011] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

[Bailleux and Boufkhad, 2003] Olivier Bailleux and Yacine Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In *International conference on principles and practice of constraint programming*, pages 108–122. Springer, 2003.

[Bard *et al.*, 2007] Gregory V Bard, Nicolas T Courtois, and Chris Jefferson. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over gf (2) via sat-solvers. *Cryptology ePrint Archive*, 2007.

[Berg *et al.*, 2024] Jeremias Berg, Matti Järvisalo, Ruben Martins, Andreas Niskanen, and Tobias Paxian. Maxsat evaluation 2024: Solver and benchmark descriptions. 2024.

[Biere and Fröhlich, 2015] Armin Biere and Andreas Fröhlich. Evaluating cdcl variable scoring schemes. In *International conference on theory and applications of satisfiability testing*, pages 405–422. Springer, 2015.

[Biere, 2013] Armin Biere. Lingeling, plingeling and treengeling entering the sat competition 2013. 2013.

[Biere, 2017] Armin Biere. CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017. In Tomáš Balyo, Marijn Heule, and Matti Järvisalo, editors, *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions*, volume B-2017-1 of *Department of Computer Science Series of Publications B*, pages 14–15. University of Helsinki, 2017.

[Cai *et al.*, 2013] Shaowei Cai, Kaile Su, and Chuan Luo. Improving walksat for random k-satisfiability problem with k $>$ 3. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27, pages 145–151, 2013.

[Cai *et al.*, 2015] Shaowei Cai, Chuan Luo, and Kaile Su. Ccanr: A configuration checking based local search solver for non-random satisfiability. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 1–8. Springer, 2015.

[Cai *et al.*, 2017] Shaowei Cai, Jinkun Lin, and Chuan Luo. Finding a small vertex cover in massive sparse graphs: Construct, local search, and preprocess. *Journal of Artificial Intelligence Research*, 59:463–494, 2017.

[Chen *et al.*, 2022] Zhihan Chen, Xindi Zhang, Shaowei Cai, and Pinyan Lu. Cdcl solvers with improved local search cooperation and pre-processing. *SAT COMPETITION*, 2022:37–38, 2022.

[Chen *et al.*, 2024] Zhihan Chen, Peng Lin, Hao Hu, and Shaowei Cai. Parls-pbo: A parallel local search solver for pseudo boolean optimization. *arXiv preprint arXiv:2407.21729*, 2024.

[Chowdhury *et al.*, 2020] Md Solimul Chowdhury, Jia You, et al. Guiding cdcl sat search via random exploration amid conflict depression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1428–1435, 2020.

[Chowdhury *et al.*, 2024] Md Solimul Chowdhury, Cayden R Codel, and Marijn JH Heule. Tassat: Transfer and share sat. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 34–42. Springer, 2024.

[Chu *et al.*, 2023] Yi Chu, Shaowei Cai, Chuan Luo, Zhendong Lei, and Cong Peng. Towards more efficient local search for pseudo-boolean optimization. In *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, pages 12–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2023.

[Codel *et al.*, 2025] Cayden Codel, Katalin Fazekas, Marijn J. H. Heule, and Markus Iser. *Proceedings of SAT Competition 2025: Solver and Benchmark Descriptions*. August 2025.

[Devriendt *et al.*, 2021] Jo Devriendt, Stephan Gocht, Emir Demirović, Jakob Nordström, and Peter J Stuckey. Cutting to the core of pseudo-boolean optimization: Combining core-guided search with cutting planes reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3750–3758, 2021.

[Elffers and others, 2020] Jan Elffers et al. A cardinal improvement to pseudo-boolean solving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1495–1503, 2020.

[Horbach, 2010] Andrei Horbach. A boolean satisfiability approach to the resource-constrained project scheduling problem. *Annals of Operations Research*, 181(1):89–107, 2010.

[Hutter *et al.*, 2002] Frank Hutter, Dave AD Tompkins, and Holger H Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In *International Conference on Principles and Practice of Constraint Programming*, pages 233–248. Springer, 2002.

[Ignatiev *et al.*, 2018] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. Pysat: A python toolkit for prototyping with sat oracles. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 428–437. Springer, 2018.

[Kovásznai *et al.*, 2018] Gergely Kovásznai, Balázs Erdélyi, and Csaba Biró. Investigations of graph properties in terms of wireless sensor network optimization. In *2018 IEEE International Conference on Future IoT Technologies (Future IoT)*, pages 1–8. IEEE, 2018.

[Kovásznai *et al.*, 2019] Gergely Kovásznai, Krisztián Gajdár, and Laura Kovács. Portfolio sat and smt solving of cardinality constraints in sensor network optimization. In *2019 21st International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 85–91. IEEE, 2019.

[Le Berre and Parrain, 2011] Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2: System description. *Journal on Satisfiability, Boolean Modelling and Computation*, 7(2-3):59–64, 2011.

[Lei *et al.*, 2020] Zhendong Lei, Shaowei Cai, and Chuan Luo. Extended conjunctive normal form and an efficient algorithm for cardinality constraints. In *IJCAI*, pages 1141–1147, 2020.

[Lei *et al.*, 2021] Zhendong Lei, Shaowei Cai, Chuan Luo, and Holger Hoos. Efficient local search for pseudo boolean optimization. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 332–348. Springer, 2021.

[Lindauer *et al.*, 2022] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

[Marques-Silva and Lynce, 2007] Joao Marques-Silva and Inês Lynce. Towards robust cnf encodings of cardinality constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 483–497. Springer, 2007.

[Martins *et al.*, 2014] Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-wbo: A modular maxsat solver. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 438–445. Springer, 2014.

[Morris, 1993] Paul Morris. The breakout method for escaping from local minima. In *Proceedings of the eleventh national conference on Artificial intelligence*, pages 40–45, 1993.

[Mull *et al.*, 2022] Nathan Mull, Shuo Pang, and Alexander Razborov. On cdcl-based proof systems with the ordered decision strategy. *SIAM Journal on Computing*, 51(4):1368–1399, 2022.

[Niemetz and Preiner, 2023] Aina Niemetz and Mathias Preiner. Bitwuzla. In *International Conference on Computer Aided Verification*, pages 3–17. Springer, 2023.

[Ogawa *et al.*, 2013] Toru Ogawa, Yangyang Liu, Ryuzo Hasegawa, Miyuki Koshimura, and Hiroshi Fujita. Modulo based cnf encoding of cardinality constraints and its application to maxsat solvers. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 9–17. IEEE, 2013.

[Reeves *et al.*, 2024] Joseph E Reeves, Marijn JH Heule, and Randal E Bryant. From clauses to klauses. In *International Conference on Computer Aided Verification*, pages 110–132. Springer, 2024.

[Reeves *et al.*, 2025] Joseph E Reeves, Joao Filipe, Min-Chien Hsu, Ruben Martins, and Marijn JH Heule. The impact of literal sorting on cardinality constraint encodings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 11327–11335, 2025.

[Rintanen *et al.*, 2006] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12-13):1031–1080, 2006.

[Selman *et al.*, 1994] Bart Selman, Henry A Kautz, Bram Cohen, et al. Noise strategies for improving local search. In *AAAI*, volume 94, pages 337–343, 1994.

[Sinz, 2005] Carsten Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. In *International conference on principles and practice of constraint programming*, pages 827–831. Springer, 2005.

[Vizel *et al.*, 2015] Yakir Vizel, Georg Weissenbacher, and Sharad Malik. Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11):2021–2035, 2015.

[Wynn, 2018] Ed Wynn. A comparison of encodings for cardinality constraints in a sat solver. *arXiv preprint arXiv:1810.12975*, 2018.

[Zhao *et al.*, 2025] Yujiao Zhao, Yiyuan Wang, Yi Chu, Wenbo Zhou, Shaowei Cai, and Minghao Yin. Improving local search algorithm for pseudo boolean optimization. *Journal of Artificial Intelligence Research*, 83, 2025.