Constraints

SAT Encoding for Set of At-Most-K Cardinality Constraints with Ladder Shape

--Manuscript Draft--

Manuscript Number:	CONS-D-25-00039
Full Title:	SAT Encoding for Set of At-Most-K Cardinality Constraints with Ladder Shape
Article Type:	Original Research
Keywords:	SAT encoding; Cardinality constraints; Sequence constraints; Sequential counter encoding; At-Most-K constraints; Anti-bandwidth
Corresponding Author:	Khanh Van To, Ph.D. VNU UET FIT: VNU University of Engineering and Technology Faculty of Information Technology Hanoi, VIET NAM
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	VNU UET FIT: VNU University of Engineering and Technology Faculty of Information Technology
Corresponding Author's Secondary Institution:	
First Author:	Hieu Xuan Truong, B.S.
First Author Secondary Information:	
Order of Authors:	Hieu Xuan Truong, B.S. Tuyen Van Kieu, B.S. Khanh Van To, Ph.D.
Order of Authors Secondary Information:	
Funding Information:	Trường Đại học Công nghệ, Đại học Quốc Gia Hà Nội (CN24.10) — Dr. Khanh Van To
Abstract:	This paper presents a novel SAT encoding designed to represent a set of AtMost-K (AMK) cardinality constraints arranged in a ladder-like structure. These constraints feature a ladder-like arrangement, with shared common variables in the center and two variables differing at the beginning and the end of the AMK constraints. The proposed encoding processes each block that contains a set of AMK constraints as a whole rather than encoding each AMK constraint individually. This approach improves the efficiency of constraint solving in SAT solvers by significantly reducing the size of the encodings, precisely the number of variables, and the generated clauses required. To achieve this, the encoding employs Sequential Counter encoding to represent each block with a ladder structure and constructs clauses that link adjacent blocks, ensuring equivalence with the input constraints. Compared to the previously proposed encoding that uses Binary Decision Diagrams for At-Most-One constraint (with k= 1), our encoding requires fewer variables and clauses, resulting in improved constraint solving times. In particular, our study is the first to develop SAT encoding for AMK constraints (with k>1) applied to the ladder constraint. Experimental results indicate that our encoding significantly enhances both the size of the encodings and the efficiency of solving time compared to existing encodings. Furthermore, we evaluated its performance on the Anti-bandwidth problem. We found that our proposed encoding surpasses alternative encodings, as well as more advanced Mixed Integer Programming (MIP) and Constraint Programming (CP) solvers. In summary, the SAT encoding technique presented in this paper provides a more efficient method for encoding sequence constraints, which could enhance the effectiveness of SAT solving for various problems involving this type of constraint.

SAT Encoding for Set of At-Most-K Cardinality Constraints with Ladder Shape

Hieu Truong Xuan[1†], Tuyen Kieu Van[1], Khanh To Van[1*†]

[1]University of Engineering and Technology,
Vietnam National University, Hanoi, Vietnam.

*Corresponding author(s). E-mail(s): khanhtv@vnu.edu.vn;
Contributing authors: truongxuanhieu11@vnu.edu.vn;
tuyenkv@vnu.edu.vn;
†These authors contributed equally to this work.

Abstract

This paper presents a novel SAT encoding designed to represent a set of At-Most-K (AMK) cardinality constraints arranged in a ladder-like structure. These constraints feature a ladder-like arrangement, with shared common variables in the center and two variables differing at the beginning and the end of the AMK constraints. The proposed encoding processes each block that contains a set of AMK constraints as a whole rather than encoding each AMK constraint individually. This approach improves the efficiency of constraint solving in SAT solvers by significantly reducing the size of the encodings, precisely the number of variables, and the generated clauses required. To achieve this, the encoding employs *Sequential Counter* encoding to represent each block with a ladder structure and constructs clauses that link adjacent blocks, ensuring equivalence with the input constraints. Compared to the previously proposed encoding that uses *Binary Decision Diagrams* for At-Most-One constraint (with $\boldsymbol{k = 1}$), our encoding requires fewer variables and clauses, resulting in improved constraint solving times. In particular, our study is the first to develop SAT encoding for AMK constraints (with $\boldsymbol{k > 1}$) applied to the ladder constraint. Experimental results indicate that our encoding significantly enhances both the size of the encodings and the efficiency of solving time compared to existing encodings. Furthermore, we evaluated its performance on the Anti-bandwidth problem. We found that our proposed encoding surpasses alternative encodings, as well as more advanced Mixed Integer Programming (MIP) and Constraint Programming (CP) solvers. In summary, the SAT encoding technique presented in this paper provides a more efficient method for encoding sequence constraints, which could enhance the effectiveness of SAT solving for various problems involving this type of constraint.

Keywords: SAT encoding, Cardinality constraints, Sequence constraints, Sequential counter encoding, At-Most-K constraints, Anti-bandwidth

1 Introduction

Sequence constraints are a type of constraint that frequently appears in combinatorial optimization problems, such as production line scheduling, employee scheduling, nurse rostering, sports timetabling, and OR problems. For instance, in the Car Sequencing Problem [1, 2], sequence constraints restrict the maximum number of cars with a specific option to no more than two in any sequence of five consecutive cars on a production line. In the nurse rostering problem within hospitals [3, 4], sequence constraints may limit a nurse to working at most four night shifts and eight evening shifts, while also requiring at least four days off within any sequence of fourteen consecutive working days [5]. These sequence constraints help ensure proper rest and recovery by preventing excessively intense work schedules. Similarly, in crew rostering, an employee may be allowed to work a maximum of five days in any seven consecutive working days [6] to comply with labor regulations. In the taxi driver shift scheduling problem at an airport for a car rental company [7], sequence constraints are used to ensure a balanced distribution of morning and afternoon shifts among drivers. In the RobinX sports timetabling problem [8], sequence constraints limit the maximum number of home or away games that a team can play within any fixed-length sequence of consecutive matches. Sequence constraints limit the number of occurrences of certain specific values in a sequence of k variables and are often referred to as AmongSeq [9] and AtMostSeq [1]. Certain problems, like the Anti-Bandwidth Problem (ABP) [10], can be reformulated to incorporate sequence constraints. ABP has applications in scheduling, radio frequency assignment, obnoxious facility location, and map coloring.

The impressive progress of SAT solvers has initiated what is now referred to as the SAT revolution [11]. These solvers are capable of addressing enormous formulas that contain millions of variables and clauses. In the near future, they are expected to handle formulas with significantly more than just millions of variables and clauses. Today, SAT solving has become a serious alternative to Integer Linear Programming (ILP) solvers for addressing NP-complete problems. Due to substantial improvements in the performance of SAT solvers, SAT solving is increasingly being used in various real-world applications. However, when applying SAT encoding to combinatorial problems, a significant number of clauses can be generated. To mitigate this issue, several encoding techniques have been proposed to reduce the overwhelming number of clauses [12, 13].

Sequence constraints have been extensively studied in the field of Constraint Programming (CP) to develop efficient solutions, as highlighted in works such as [5] and [2]. However, effective SAT encodings for sequence constraints, like AmongSeq [9] and AtMostSeq [1], remain largely unexamined. A notable exception is the *Duplex* encoding [14], which is specifically designed for At-Most-One (AMO) constraints. In our research, we propose a block-based encoding method for a collection of At-Most-K

(AMK) constraints, rather than encoding each AMK constraint independently within the sequence constraints. These blocks are encoded using the *Sequential Counter* encoding [15], which incorporates auxiliary variables to connect adjacent blocks. This approach not only demonstrates greater efficiency for AMO constraints when compared to the *Duplex* encoding based on *Binary Decision Diagrams* [16], but also proves effective for AMK constraints when $k > 1$.

In summary, the contributions of this paper are as follows:

- *We propose a novel SAT encoding for a set of AMK cardinality constraints arranged in a ladder shape. Our approach leverages shared variables among neighboring AMK constraints, allowing us to organize them into blocks for encoding rather than encoding each constraint independently. To the best of our knowledge, this is the first SAT encoding method for AMK cardinality constraints with a ladder shape where $k > 1$.*
- *Additionally, we present a solution to the Anti-bandwidth problem using our new SAT encoding for At-most-one cardinality constraints. Our results show that our approach outperforms existing SAT encodings, as well as Mixed Integer Programming (MIP) and Constraint Programming (CP) methods that utilize powerful solvers such as CPLEX and Gurobi.*

This paper extends our previously published research [16] in the following ways: First, we generalize the encoding from AMO constraints to AMK constraints with $k > 1$ for sets of AMK cardinality constraints in a ladder shape, whereas the previous work focused solely on AMO constraints ($k = 1$). Second, for the Anti-bandwidth problem, we develop a parallel search implementation, as opposed to the sequential approach used in [16], leading to improved performance. Third, we broaden our experimental work to include comparisons and evaluations with powerful MIP and CP solvers, such as *Gurobi* and Google's *CP-SAT*. This comparison highlights the effectiveness of our proposed SAT encoding in solving Anti-bandwidth problems when compared to MIP and CP solvers. Additionally, this paper presents the experiments in more detail, including thorough evaluations and discussions.

The paper is organized as follows: Section 1 introduces the research problem and summarizes the contributions of this study. Section 2 presents the basic definitions of the Ladder AMK constraint and discusses how to represent one AMK constraint using two sub-AMK constraints. Section 3 details the block encoding for the Ladder AMK constraint and includes a comparative evaluation against existing SAT encodings. Section 4 introduces the encoding for the Ladder AMO constraint, which is a specific case of the Ladder AMK constraint with $k = 1$. Section 5 explains the approach to solving the Anti-bandwidth problem using SAT solving. Section 6 provides experimental results that compare our proposed encoding to existing SAT encodings, as well as MIP and CP solvers, and presents the experimental results for the Anti-bandwidth problem. Finally, Section 7 concludes the paper by discussing the applications of the proposed SAT encoding techniques in scheduling problems.

2 Preliminaries

2.1 Ladder AMK constraint

A Ladder AMK constraint consists of a set of AMK cardinality constraints that involve w boolean variables. In this structure, each pair of adjacent AMK constraints shares $w-1$ variables. The only differences between the two adjacent AMK constraints are the first variable of the first constraint and the last variable of the second constraint. This overlapping feature results in a ladder-like shape. The formal definition of the Ladder AMK constraint is provided in Definition 1 below:

Definition 1. *Given a sequence of n boolean variables $\Omega = \{x_1, x_2, ..., x_n\}$ and a number w such that $1 < w \leq n$, a Ladder AMK constraint of width w is formulated as follows:*

$$Ladder(\Omega, w) = \bigwedge_{i=0}^{n-w} \Big(\sum_{j=i+1}^{i+w} x_j \leq k \Big) \qquad 1 \leq k < w$$

Example 1. *Given a sequence of 10 boolean variables $\Omega = \{x_1, x_2, ..., x_{10}\}$, the Ladder AMK constraint of width $w = 4$ can be illustrated in Figure 1 as follows:*

$$\begin{array}{cccccccccccccccccccc}
x_1 & + & x_2 & + & x_3 & + & x_4 & & & & & & & & & & & & & \leq k \wedge \\
 & & x_2 & + & x_3 & + & x_4 & + & x_5 & & & & & & & & & & & \leq k \wedge \\
 & & & & x_3 & + & x_4 & + & x_5 & + & x_6 & & & & & & & & & \leq k \wedge \\
 & & & & & & x_4 & + & x_5 & + & x_6 & + & x_7 & & & & & & & \leq k \wedge \\
 & & & & & & & & x_5 & + & x_6 & + & x_7 & + & x_8 & & & & & \leq k \wedge \\
 & & & & & & & & & & x_6 & + & x_7 & + & x_8 & + & x_9 & & & \leq k \wedge \\
 & & & & & & & & & & & & x_7 & + & x_8 & + & x_9 & + & x_{10} & \leq k
\end{array}$$

Fig. 1: Ladder AMK constraint of 10 variables and width 4.

Figure 1 illustrates a Ladder AMK constraint consisting of 7 AMK constraints, each with a width of 4. The difference between the first variable of the preceding constraint and the last variable of the subsequent constraint forms a ladder structure. When $k = 1$, Example 1 represents the Ladder AMO constraint, where at most one boolean variable is true in each of the 7 AMO constraints.

2.2 Decomposition of AMK constraint

An AMK constraint can be reformulated as two separate AMK constraints applied to two subsets of the original set of variables. Proposition 1 explains how to decompose an AMK constraint involving n variables by dividing it into two sub-AMK constraints. This decomposition incorporates connecting clauses to ensure that the partitioning complies with the original constraint without violating it.

Proposition 1. *The constraint* $x_1+x_2+\ldots+x_n \leq k$ *holds iff for all* i *s.t.* $1 \leq i < n$ *:*

$$(x_1 + \ldots + x_i \leq k) \wedge (x_{i+1} + \ldots + x_n \leq k)$$

$$\wedge \bigwedge_{j=1}^{k} (x_1 + \ldots + x_i \leq k - j \vee x_{i+1} + \ldots + x_n \leq j - 1)$$

Proposition 1 can be easily proven to be correct by considering all possible cases in which one of the two sub-expressions has totals of $k, k-1, k-2, \ldots, 1$. In each of these cases, the other sub-expression must have a corresponding total that is less than or equal to $0, 1, 2, \ldots, k-1$. The following example illustrates the decomposition of an AMK constraint with $k = 3$ over 8 variables $\{x_1, \ldots, x_8\}$ as follows:

$$\begin{aligned} \sum_{i=1}^{8} x_i \leq 3 \quad \equiv \quad & (x_1 + x_2 + x_3 + x_4 \leq 3) \wedge (x_5 + x_6 + x_7 + x_8 \leq 3) \\ & \wedge((x_1 + x_2 + x_3 + x_4 \leq 2) \vee (x_5 + x_6 + x_7 + x_8 \leq 0)) \\ & \wedge((x_1 + x_2 + x_3 + x_4 \leq 1) \vee (x_5 + x_6 + x_7 + x_8 \leq 1)) \\ & \wedge((x_1 + x_2 + x_3 + x_4 \leq 0) \vee (x_5 + x_6 + x_7 + x_8 \leq 2)), \end{aligned}$$

The Ladder AMO constraint is a specific instance of the Ladder AMK constraint, which occurs when $k = 1$. To derive the decomposition of the AMO constraint, one can simply substitute k with 1 in the decomposition of the AMK constraint. Lemma 1 provides a detailed explanation of how to decompose an AMO constraint that involves n variables.

Lemma 1. *The constraint* $x_1 + x_2 + \ldots + x_n \leq 1$ *holds iff for all* i *s.t.* $1 \leq i < n$*:*

$$(x_1 + \ldots + x_i \leq 1) \wedge (x_{i+1} + \ldots + x_n \leq 1) \wedge (x_1 + \ldots + x_i \leq 0 \vee x_{i+1} + \ldots + x_n \leq 0)$$

3 SCL encoding for Ladder AMK constraint

3.1 Partition the Ladder AMK constraint into sub-blocks

Our encoding approach leverages shared variables among neighboring AMK constraints to organize them into blocks, rather than encoding each AMK constraint independently. The Ladder AMK constraint is divided into sub-blocks based on AMK decomposition as described in Section 2.2. Each block is then encoded using *Sequential Counter* (*Seq*) encoding [15]. The required auxiliary variables for block encoding are incorporated into clauses that connect two adjacent blocks (Proposition 1).

In the context of a Ladder AMK constraint, we start by dividing its set of variables $\Omega = \langle x_1, x_2, \ldots, x_n \rangle$ into $M = \lceil \frac{n}{w} \rceil$ consecutive, non-overlapping subsets, each consisting of w variables. It is important to note that if n is not divisible by w, the last subset may contain fewer variables. Next, using these subset divisions as a framework, we partition the original AMK constraints into groups that consist of nested sub-expressions. Owing to the nested nature of these expressions, the resulting groups create discrete blocks that effectively encapsulate the exclusivity variables associated

with their corresponding subset. Figure 2 illustrates the process of decomposing and generating blocks for the Ladder AMK constraint, which is detailed in Example 1. The set of variables in this Ladder AMK constraint is divided into three non-overlapping subsets. The first two subsets, each matching the size of the AMK constraints, are $\{x_1, x_2, x_3, x_4\}$ and $\{x_5, x_6, x_7, x_8\}$. The last subset contains the remaining variables, $\{x_9, x_{10}\}$. Subsequently, these AMK constraints are decomposed and grouped into blocks, as depicted in the framed sections of Figure 2.

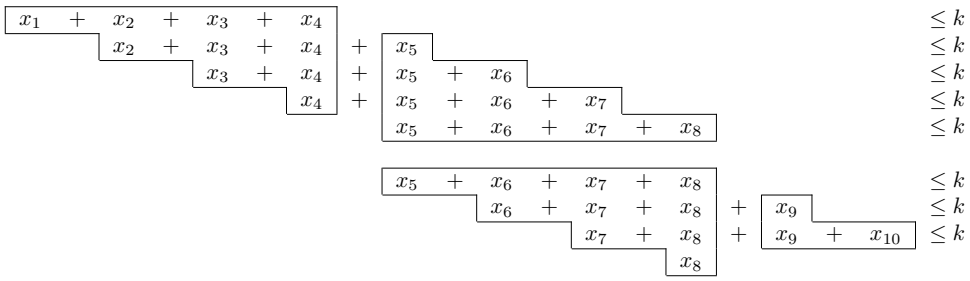

Fig. 2: Decomposition of Ladder AMK constraint.

3.2 Encoding for Ladder AMK constraint

In this section, we introduce our proposed encoding called ***SCL*** (*Sequential Counter encoding for Ladder constraints*) for encoding Ladder AMK constraints. The *SCL* utilizes a breakdown process to decompose a Ladder AMK constraint into discrete blocks containing nested sub-expressions, which are then encoded using a single *Sequential Counter* (*Seq*) [15]. The register bits produced by the *Seq* encoding enhance the connection between adjacent blocks, effectively reformulating the original constraints.

Given a Ladder AMK constraint with width w over n variables $\Omega = \{x_1, x_2, \ldots, x_n\}$, we first divide Ω into $M = \lceil \frac{n}{w} \rceil$ subsets $\{\omega_1, \ldots, \omega_M\}$. Each subset ω_i contains up to w unique consecutive variables s.t. $w_i = \{x_{i,1}, \ldots, x_{i,w}\}$. To represent the subset ω_i, we use two AMK *Seq* blocks to encode the constraint $(x_{i,1} + \ldots + x_{i,w} \leq k)$: the first *Seq* orders the variables from left to right (from $x_{i,1}$ to $x_{i,w}$), while the second *Seq* orders them in the reverse direction (from $x_{i,w}$ to $x_{i,1}$).

Consider the Ladder AMK constraint in Figure 2 as an example. The set of variables $\Omega = \{x_1, x_2, \ldots, x_{10}\}$ is divided into three non-overlapping subsets $\omega_1 = \{x_1, x_2, x_3, x_4\}$, $\omega_2 = \{x_5, x_6, x_7, x_8\}$, and $\omega_3 = \{x_9, x_{10}\}$. *Seq* constructing ω_1 with right-to-left variable ordering produces four sub-expressions: $\{x_4\}$, $\{x_3 + x_4\}$, $\{x_2 + x_3 + x_4\}$ and $\{x_1 + x_2 + x_3 + x_4\}$. Meanwhile, applying *Seq* construction to ω_2 in reverse ordering yields another four sub-expressions: $\{x_5\}$, $\{x_5 + x_6\}$, $\{x_5 + x_6 + x_7\}$, and $\{x_5 + x_6 + x_7 + x_8\}$. These sub-expressions can then be connected to reformulate the original AMK constraints. Specifically, connecting $\{x_2 + x_3 + x_4\}$ and $\{x_5\}$ results in $\{x_2 + x_3 + x_4 + x_5\}$, connecting $\{x_3 + x_4\}$ and $\{x_5 + x_6\}$ yields $\{x_3 + x_4 + x_5 + x_6\}$, and so on for other combinations.

a. ***Encoding each block in the Ladder AMK constraint***

We denote $x_{i,j}$ as the j^{th} element in the sequence of variables for the block i. Since $x_{i,j}$ is binary, the sum $x_{i,1} + \ldots + x_{i,j}$ can take any integer value from 0 to j. To represent this sum, j register bits are required, denoted as $R_{i,j,s}$ for $s \in \{1, \ldots, j\}$. Each register bit $R_{i,j,s}$ indicates that under the condition "the sum of the first j variables in block i is greater than or equal to s". In other words, $R_{i,j,s}$ is set to true if and only if $x_{i,1} + \cdots + x_{i,j} \geq s$, and false otherwise, which means $x_{i,1} + \cdots + x_{i,j} \leq s - 1$.

It is necessary to mention that our objective is to satisfy the At-Most-K constraint. Consequently, when representing the sum $x_{i,1} + \ldots + x_{i,j}$ under the condition $\leq k$, if $k < j$, it suffices to use only k register bits. For example, to represent $x_{1,1} + x_{1,2} + x_{1,3}$ under the At-Most-Two condition ($k = 2$), we only require two register bits $R_{1,3,1}$ and $R_{1,3,2}$.

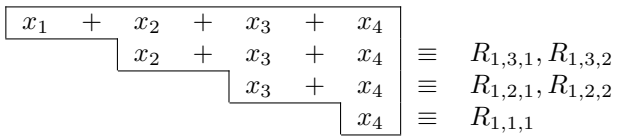

Fig. 3: Register bit construction of block $x_1 + x_2 + x_3 + x_4$ under the condition ≤ 2.

Figure 3 illustrates the assignment of register bits for the first block (the leftmost block) of the Ladder constraint, as presented in Figure 2, under the condition ≤ 2. The sub-expression $\{x_4\}$ has a maximum value of 1, and thus requires only one register bit, denoted $R_{1,1,1}$. Next, the sub-expression $\{x_3 + x_4\}$ has a maximum value of 2, and therefore is assigned two register bits, $R_{1,2,1}$ and $R_{1,2,2}$. For $\{x_2 + x_3 + x_4\}$, although the maximum value is 3, due to the constraint ≤ 2, this sub-expression is also assigned only two register bits, namely $R_{1,3,1}$ and $R_{1,3,2}$. Finally, the sub-expression $\{x_1 + x_2 + x_3 + x_4\}$ does not require additional register bits, since condition ≤ 2 can be expressed based on the value of x_1 and the register bits already assigned to $x_2 + x_3 + x_4$.

We also note that, based on the indication of $R_{i,j,s}$, if $R_{i,j,s}$ is *true*, then $R_{i,j,s-1}$ must also be *true*, since $x_{i,1} + \ldots + x_{i,j} \geq s$ implies $x_{i,1} + \ldots + x_{i,j} \geq s - 1$. Furthermore, the equality $x_{i,1} + \ldots + x_{i,j} = s$ is valid if and only if $R_{i,j,s}$ is *true* and $R_{i,j,s+1}$ is $false$. Specifically, $R_{i,j,s}$ is *true* indicating that $x_{i,1} + \ldots + x_{i,j} \geq s$, while $R_{i,j,s+1}$ is $false$ implying that $x_{i,1} + \ldots + x_{i,j} \leq (s+1) - 1 = s$. Combining these two conditions yields $s \leq x_{i,1} + \ldots + x_{i,j} \leq s$, and thus $x_{i,1} + \ldots + x_{i,j} = s$.

Using the denoted variables $x_{i,j}$ and register bits $R_{i,j,s}$, the AMK *Seq* construction for blocks involves the application of seven formulas 1, 2, 3, 4, 5, 6, and 7 as detailed below.

$$\bigwedge_{j=1}^{w-1} x_{i,j} \rightarrow R_{i,j,1} \tag{1}$$

$$\bigwedge_{j=2}^{w-1} \bigwedge_{s=1}^{min(j-1,k)} R_{i,j-1,s} \rightarrow R_{i,j,s} \tag{2}$$

$$\bigwedge_{j=2}^{w-1} \bigwedge_{s=2}^{min(j,k)} x_{i,j} \wedge R_{i,j-1,s-1} \rightarrow R_{i,j,s} \tag{3}$$

$$\bigwedge_{j=1}^{k} \neg x_{i,j} \rightarrow \neg R_{i,j,j} \tag{4}$$

$$\bigwedge_{j=2}^{w-1} \bigwedge_{s=2}^{min(j,k)} \neg R_{i,j-1,s-1} \rightarrow \neg R_{i,j,s} \tag{5}$$

$$\bigwedge_{j=2}^{w-1} \bigwedge_{s=1}^{min(j-1,k)} \neg x_{i,j} \wedge \neg R_{i,j-1,s} \rightarrow \neg R_{i,j,s} \tag{6}$$

$$\bigwedge_{j=k+1}^{w} x_{i,j} \rightarrow \neg R_{i,j-1,k} \tag{7}$$

Formula 1 specifies that the sum of the first j variables in block i must be at least one if $x_{i,j}$ is $true$. Formula 2 states that if the sum of the first $j-1$ variables in block i is greater than or equal to s, then the sum of the first j variables must also be at least s. Formula 3 ensures that if $x_{i,j}$ is $true$ and the sum of the first $j-1$ variables is greater than or equal to $s-1$, then the sum of the first j variables in block i must be at least s. Together, formulas 1, 2, and 3 ensure the correctness of the statement "$R_{i,j,s}$ holds $true$ if and only if $x_{i,1} + \ldots + x_{i,j} \geq s$".

Next, formula 4 limits the sum of the first j variables in block i to at most $j-1$ if $x_{i,j}$ is $false$. The formula 5 states that if the sum of the first $j-1$ variables is less than $s-1$, then the sum of the first j variables must be less than s. Formula 6 states that if $x_{i,j}$ is $false$ and the sum of the first $j-1$ variables is less than s, then the sum of the first j variables must also be less than s. Taken together, the formulas 4, 5, and 6 validate the correctness of the statement "$R_{i,j,s}$ is $false$ if and only if $x_{i,1} + \ldots + x_{i,j} \leq s-1$".

Finally, the formula 7 stipulates that if $x_{i,j}$ is true, then the sum of the first $j-1$ variables in block i must be strictly less than k. By applying this constraint to all indices j within the range of $[k+1, w]$, we ensure that no more than k out of the w variables can be set to true. Additionally, the sizes of the clauses (i.e., the number of literals in each clause) in our encoding are at most 3.

It is necessary to note that when two blocks both encapsulate the same subset of variables, if one block satisfies the AMK property over that subset using formula 7, the other block will also inherently satisfy the AMK condition without requiring the direct application of the same formula. Leveraging this insight, when encoding two different blocks within the same subset, we impose the AMK constraint on only one of them. We refer to the block that enforces the AMK constraint as the

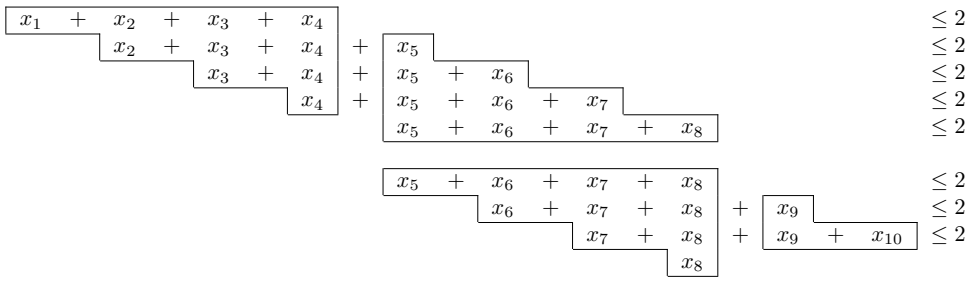

Fig. 4: Decomposition of Ladder At-Most-Two constraint.

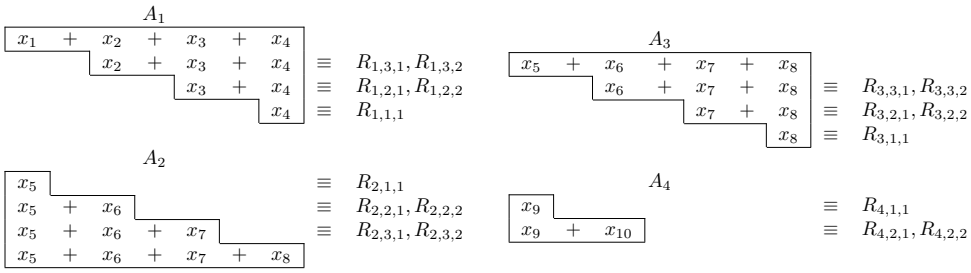

Fig. 5: Register bits constructing of Ladder At-Most-Two constraint.

AMK block, while the other block, which primarily provides sub-expressions to connect adjacent blocks, is known as the connection block.

Figure 4 illustrates an example of a Ladder AMK constraint with $k = 2$. This Ladder AMK constraint is divided into four blocks, labeled A_1, A_2, A_3, and A_4. Figure 5 shows these four blocks along with the necessary register bits used in the *Seq* encoding. The block A_1, representing the constraint $x_1 + x_2 + x_3 + x_4 \leq 2$, is encoded using *Seq* with the formulas from equations 1 to 7. The generated clauses for constructing block A_1 are presented below. In this context, the variables are defined as follows: $x_{1,1} \equiv x_4$, $x_{1,2} \equiv x_3$, $x_{1,3} \equiv x_2$, $x_{1,4} \equiv x_1$, and $x_4 \equiv R_{1,1,1}$.

$$
\begin{array}{lcl}
(1) & \overset{A_1}{\Longleftrightarrow} & \begin{array}{lcl} x_{1,2} & \rightarrow & R_{1,2,1} \\ x_{1,3} & \rightarrow & R_{1,3,1} \end{array} \iff \begin{array}{lcl} x_3 & \rightarrow & R_{1,2,1} \\ x_2 & \rightarrow & R_{1,3,1} \end{array} \\[2ex]
(2) & \overset{A_1}{\Longleftrightarrow} & \begin{array}{lcl} R_{1,1,1} & \rightarrow & R_{1,2,1} \\ R_{1,2,1} & \rightarrow & R_{1,3,1} \\ R_{1,2,2} & \rightarrow & R_{1,3,2} \end{array} \iff \begin{array}{lcl} x_4 & \rightarrow & R_{1,2,1} \\ R_{1,2,1} & \rightarrow & R_{1,3,1} \\ R_{1,2,2} & \rightarrow & R_{1,3,2} \end{array} \\[2ex]
(3) & \overset{A_1}{\Longleftrightarrow} & \begin{array}{lcl} x_{1,2} \wedge R_{1,1,1} & \rightarrow & R_{1,2,2} \\ x_{1.3} \wedge R_{1,2,1} & \rightarrow & R_{1,3,2} \end{array} \iff \begin{array}{lcl} x_3 \wedge x_4 & \rightarrow & R_{1,2,2} \\ x_2 \wedge R_{1,2,1} & \rightarrow & R_{1,3,2} \end{array} \\[2ex]
(4) & \overset{A_1}{\Longleftrightarrow} & \neg x_{1,2} \rightarrow \neg R_{1,2,2} \iff \neg x_3 \rightarrow \neg R_{1,2,2}
\end{array}
$$

$$
\begin{array}{lcccccccc}
(5) & \overset{A_1}{\Longleftrightarrow} & \neg R_{1,1,1} & \rightarrow & \neg R_{1,2,2} & \Longleftrightarrow & \neg x_4 & \rightarrow & \neg R_{1,2,2} \\
 & & \neg R_{1,2,1} & \rightarrow & \neg R_{1,3,2} & & \neg R_{1,2,1} & \rightarrow & \neg R_{1,3,2} \\
(6) & \overset{A_1}{\Longleftrightarrow} & \neg x_{1,2} \wedge \neg R_{1,1,1} & \rightarrow & \neg R_{1,2,1} & & \neg x_3 \wedge \neg x_4 & \rightarrow & \neg R_{1,2,1} \\
 & & \neg x_{1,3} \wedge \neg R_{1,2,1} & \rightarrow & \neg R_{1,3,1} & \Longleftrightarrow & \neg x_2 \wedge \neg R_{1,2,1} & \rightarrow & \neg R_{1,3,1} \\
 & & \neg x_{1,3} \wedge \neg R_{1,2,2} & \rightarrow & \neg R_{1,3,2} & & \neg x_2 \wedge \neg R_{1,2,2} & \rightarrow & \neg R_{1,3,2} \\
(7) & \overset{A_1}{\Longleftrightarrow} & x_{1,3} & \rightarrow & \neg R_{1,2,2} & \Longleftrightarrow & x_2 & \rightarrow & \neg R_{1,2,2} \\
 & & x_{1,4} & \rightarrow & \neg R_{1,3,2} & & x_1 & \rightarrow & \neg R_{1,3,2}
\end{array}
$$

Building the block A_2 is the same as A_1, including the following clauses generated by *Seq* encoding:

$$
\begin{array}{lcccccccc}
(1) & \overset{A_2}{\Longleftrightarrow} & x_{2,2} & \rightarrow & R_{2,2,1} & \Longleftrightarrow & x_6 & \rightarrow & R_{2,2,1} \\
 & & x_{2,3} & \rightarrow & R_{2,3,1} & & x_7 & \rightarrow & R_{2,3,1} \\
(2) & \overset{A_2}{\Longleftrightarrow} & R_{2,1,1} & \rightarrow & R_{2,2,1} & & x_5 & \rightarrow & R_{2,2,1} \\
 & & R_{2,2,1} & \rightarrow & R_{2,3,1} & \Longleftrightarrow & R_{2,2,1} & \rightarrow & R_{2,3,1} \\
 & & R_{2,2,2} & \rightarrow & R_{2,3,2} & & R_{2,2,2} & \rightarrow & R_{2,3,2} \\
(3) & \overset{A_2}{\Longleftrightarrow} & x_{2,2} \wedge R_{2,1,1} & \rightarrow & R_{2,2,2} & \Longleftrightarrow & x_6 \wedge x_5 & \rightarrow & R_{2,2,2} \\
 & & x_{2.3} \wedge R_{2,2,1} & \rightarrow & R_{2,3,2} & & x_7 \wedge R_{2,2,1} & \rightarrow & R_{2,3,2} \\
(4) & \overset{A_2}{\Longleftrightarrow} & \neg x_{2,2} & \rightarrow & \neg R_{2,2,2} & \Longleftrightarrow & \neg x_6 & \rightarrow & \neg R_{2,2,2} \\
(5) & \overset{A_2}{\Longleftrightarrow} & \neg R_{2,1,1} & \rightarrow & \neg R_{2,2,2} & \Longleftrightarrow & \neg x_5 & \rightarrow & \neg R_{2,2,2} \\
 & & \neg R_{2,2,1} & \rightarrow & \neg R_{2,3,2} & & \neg R_{2,2,1} & \rightarrow & \neg R_{2,3,2} \\
(6) & \overset{A_2}{\Longleftrightarrow} & \neg x_{2,2} \wedge \neg R_{2,1,1} & \rightarrow & \neg R_{2,2,1} & & \neg x_6 \wedge \neg x_5 & \rightarrow & \neg R_{2,2,1} \\
 & & \neg x_{2,3} \wedge \neg R_{2,2,1} & \rightarrow & \neg R_{2,3,1} & \Longleftrightarrow & \neg x_7 \wedge \neg R_{2,2,1} & \rightarrow & \neg R_{2,3,1} \\
 & & \neg x_{2,3} \wedge \neg R_{2,2,2} & \rightarrow & \neg R_{2,3,2} & & \neg x_7 \wedge \neg R_{2,2,2} & \rightarrow & \neg R_{2,3,2} \\
(7) & \overset{A_2}{\Longleftrightarrow} & x_{2,3} & \rightarrow & \neg R_{2,2,2} & \Longleftrightarrow & x_7 & \rightarrow & \neg R_{2,2,2} \\
 & & x_{2,4} & \rightarrow & \neg R_{2,3,2} & & x_8 & \rightarrow & \neg R_{2,3,2}
\end{array}
$$

Since block A_2 has already been designed to encapsulate the constraint $x_5 + x_6 + x_7 + x_8 \leq 2$, it is redundant for block A_3 to reiterate this constraint. However, block A_3 will leverage the formulas from equations 1 to 6 to generate the necessary register bits for its connection to the adjacent block, A_4. This approach stems from the AMK decomposition. It is essential to note that equation 7 can be applied in A_3, provided that block A_2 has not been utilized. Similarly, block A_4 encodes the following clauses: $(x_9 \rightarrow R_{4,2,1}) \wedge (x_{10} \rightarrow R_{4,2,1}) \wedge ((x_9 \wedge x_{10}) \rightarrow R_{4,2,2}) \wedge (\neg x_9 \rightarrow \neg R_{4,2,2}) \wedge (\neg x_{10} \rightarrow \neg R_{4,2,2}) \wedge ((\neg x_9 \wedge \neg x_{10}) \rightarrow \neg R_{4,2,1})$.

b. ***Clauses connecting two adjacent blocks***

The blocks are formed by applying the AMK decomposition according to Proposition 1. Therefore, after performing the At-Most-K encoding for each block, it is necessary to construct clauses to connect two adjacent blocks formed by the AMK decomposition. Blocks A_1 and A_2 are formed through the AMK decomposition as follows:

$$(x_2 + x_3 + x_4 + x_5 \leq 2) \overset{\text{Prop. 1}}{\equiv} \begin{array}{l}(x_2 + x_3 + x_4 \leq 2) \wedge (x_5 \leq 2)\\ \wedge(x_2 + x_3 + x_4 \leq 1 \vee x_5 \leq 0)\\ \wedge(x_2 + x_3 + x_4 \leq 0 \vee x_5 \leq 1)\end{array}$$

$$(x_3 + x_4 + x_5 + x_6 \leq 2) \overset{\text{Prop. 1}}{\equiv} \begin{array}{l}(x_3 + x_4 \leq 2) \wedge (x_5 + x_6 \leq 2)\\ \wedge(x_3 + x_4 \leq 1 \vee x_5 + x_6 \leq 0)\\ \wedge(x_3 + x_4 \leq 0 \vee x_5 + x_6 \leq 1)\end{array}$$

$$(x_4 + x_5 + x_6 + x_7 \leq 2) \overset{\text{Prop. 1}}{\equiv} \begin{array}{l}(x_4 \leq 2) \wedge (x_5 + x_6 + x_7 \leq 2)\\ \wedge(x_4 \leq 1 \vee x_5 + x_6 + x_7 \leq 0)\\ \wedge(x_4 \leq 0 \vee x_5 + x_6 + x_7 \leq 1)\end{array}$$

Consequently, the connection between block A_1 and block A_2 can be represented by a set of clauses as follows:

$$\begin{array}{c}(x_2 + x_3 + x_4 \leq 1 \vee x_5 \leq 0) \wedge (x_2 + x_3 + x_4 \leq 0 \vee x_5 \leq 1)\\ \wedge(x_3 + x_4 \leq 1 \vee x_5 + x_6 \leq 0) \wedge (x_3 + x_4 \leq 0 \vee x_5 + x_6 \leq 1)\\ \wedge(x_4 \leq 1 \vee x_5 + x_6 + x_7 \leq 0) \wedge (x_4 \leq 0 \vee x_5 + x_6 + x_7 \leq 1)\end{array}$$

We utilize the corresponding register bits to express the connection between A_1 and A_2. For instance, the expression $x_2 + x_3 + x_4 \leq 1$ serves as the negation of $x_2 + x_3 + x_4 \geq 2$ and can be replaced with $\neg R_{1,3,2}$. Similarly, the expression $x_2 + x_3 + x_4 \leq 0$ is the negation of $x_2 + x_3 + x_4 \geq 1$ and can be represented as $\neg R_{1,3,1}$. This substitution can be applied to the remaining sub-expressions as well. Therefore, the connection between block A_1 and block A_2 can be expressed more straightforwardly as follows:

$$\begin{array}{c}(\neg R_{1,3,2} \vee x_5 \leq 0) \wedge (\neg R_{1,3,1} \vee x_5 \leq 1)\\ \wedge(\neg R_{1,2,2} \vee \neg R_{2,2,1}) \wedge (\neg R_{1,2,1} \vee \neg R_{2,2,2})\\ \wedge(x_4 \leq 1 \vee \neg R_{2,3,1}) \wedge (x_4 \leq 0 \vee \neg R_{2,3,2})\end{array}$$

In summary, connecting two consecutive blocks i and $i+1$ in the Ladder AMK constraint involves a series of disjunctions, as shown in formula 8. This formula can also be optimized by reusing register bits from the *Seq* encoding of the individual blocks.

$$\bigwedge_{j=2}^{w} \bigwedge_{p=1}^{k} \Big(\sum_{q=1}^{w-j+1} x_{i,q} \leq k - p \vee \sum_{q'=1}^{j-1} x_{i+1,q'} \leq p - 1 \Big) \tag{8}$$

3.3 Comparison of Ladder AMK constraint encodings

In this section, we assess the performance of the *SCL* encoding on Ladder AMK constraints ($SCL_{\leq k}$) and compare it with other encodings that encode each AMK

constraint individually. These encodings include *Pairwise*, *Sequential Counter* (*Seq*) [15], *Binary Decision Diagrams* (*BDD*) [17], and *Cardinality Network* (*Card*) [18, 19].

Table 1: Size of SAT encoding for a Ladder AMK constraint over n variables with width w, $N = (n - w) + 1$ and $M = \lceil \frac{n}{w} \rceil$.

Encoding	Technique	Auxiliary variables	Clauses	Complexity
Pairwise	Encode each AMK constraint separately	0	$C_{k+1}^{w} + (N-1)C_{k}^{w-1}$	$\mathcal{O}(n^{k+1})$
Seq		$Nk(w-2)$	$N(3kw - 7k + w)$	$\mathcal{O}(n^2k)$
BDD		$N(w-k)(k+1)$	$2N(w-k)(k+1)$	$\mathcal{O}(n^2k)$
Card		$\mathcal{O}(n^2\log_2^2 k)$	$\mathcal{O}(n^2\log_2^2 k)$	$\mathcal{O}(n^2\log_2^2 k)$
$SCL_{\leq k}$	Encode set of AMK constraints	$(2M-2)(wk - \frac{k^2+k}{2} - 1)$	$9Mkw - 5Mk^2 - Mw - 7Mk - 9kw + 5k^2 - 2M + 2w + 6k + 2$	$\mathcal{O}(nk)$

Table 1 summarizes the evaluation results of different encodings used to represent Ladder AMK constraints over n variables with a width w. The encodings are compared based on the number of required auxiliary variables, the number of generated clauses, and the complexity of the clause space in the worst case, specifically when w is approximately $\frac{n}{2}$.

Pairwise encoding generates a clause for every set of $k+1$ variables among the w variables in each AMK constraint. These clauses ensure that at least one variable in any set of $k+1$ variables is assigned a value of false. This means that no more than k variables can be assigned the value true. By eliminating redundant constraints, *Pairwise* encoding uses C_{k+1}^{w} clauses to encode the first AMK constraint and C_{k}^{w-1} clauses to encode each of the remaining $N-1$ AMK constraints. As a result, the total number of clauses is $C_{k+1}^{w} + (N-1)C_{k}^{w-1}$, leading to a complexity of $\mathcal{O}(n^{k+1})$.

Sequential Counter (*Seq*) encoding [15] employs a sequential counting technique. In the context of the AMK constraint, this encoding ensures that in any sequence of w consecutive variables, no more than k variables can be assigned *true*, thereby maintaining the AMK property. Each *Seq* encoding for an AMK constraint over w variables requires at most $k(w-2)$ additional variables and generates approximately $3kw-7k+w$ clauses, leading to a total of $Nk(w-2)$ auxiliary variables, $N(3kw-7k+w)$ clauses, and a complexity of $\mathcal{O}(n^2k)$.

Binary Decision Diagrams (*BDD*) encoding [17] represents the AMK constraint as a circuit of *if-then-else* (*ITE*) gates. The goal of this encoding is to constrain the circuit's inputs to ensure that signal propagation through the gates ultimately reaches the final gate. For an AMK constraint on w variables, *BDD* encoding constructs a circuit with approximately $(w-k)(k+1)$ gates, corresponding to the same number of auxiliary variables. Each gate in the circuit has two outputs, representing the boolean values *true* and *false*, leading to a total of approximately $2(w-k)(k+1)$ clauses. In summary, encoding a Ladder AMK constraint using the *BDD* encoding requires approximately $N(w-k)(k+1)$ auxiliary variables, $2N(w-k)(k+1)$ clauses, and has a complexity of $\mathcal{O}(n^2k)$.

Leveraging the odd-even merging technique in *Sorting Network* encoding [20], Asín et al. introduced the *Half Sorting Network* and *Half Merging Network* encodings [19], which reduce the number of clauses by half. Building on these approaches, Asín et al. further proposed the *Simplified Half-Merging Network*, a streamlined version of the *Half Merging Network*. This encoding later served as the foundation for the development of *Cardinality Network* (*Card*) encoding [19]. For an AMK constraint of w variables, *Card* encoding uses $\mathcal{O}(n\log_2^2 k)$ auxiliary variables and $\mathcal{O}(n\log_2^2 k)$ clauses. When applied to Ladder AMK constraints, this encoding requires a total of $\mathcal{O}(n^2\log_2^2 k)$ auxiliary variables, $\mathcal{O}(n^2\log_2^2 k)$ clauses, and has a complexity of $\mathcal{O}(n^2\log_2^2 k)$.

Our analysis reveals that the $SCL_{\leq k}$ encoding results in fewer clauses than other encodings. For simplicity, we assume that the size of the last subset matches that of the others, although it is typically smaller. Since only one *Seq* is required for the first and last subsets, we construct one AMK block for each. For the remaining $(M-2)$ subsets, which need two SCs oriented in opposite directions, we generate a total of $(M-2)$ AMK blocks and $(M-2)$ connection blocks. It is important to note that in our encoding, each AMK block requires $4kw - 2k^2 - 4k - 1$ clauses, whereas each connection block utilizes $(w-k)$ fewer clauses than the AMK block. Furthermore, connecting these $2+(M-2)+(M-2) = 2M-2$ blocks necessitates $M-1$ connections, with each connection requiring $(w-k)k$ clauses. In conclusion, the total number of clauses employed by the $SCL_{\leq k}$ encoding is:

AMK-blocks-clauses	=	$2+(M-2)$	x	$(4kw - 2k^2 - 4k - 1)$
Connection-blocks-clauses	=	$(M-2)$	x	$(4kw - 2k^2 - 4k - 1 - w + k)$
Connect-clauses	=	$(M-1)$	x	$(w-k)k$
Total-clauses	=	$9Mkw - 5Mk^2 - Mw - 7Mk - 9kw + 5k^2 - 2M + 2w + 6k + 2$		

In the calculation of auxiliary variable usage, $SCL_{\leq k}$ encoding employs $wk - \frac{k^2+k}{2}$ register bits per block, including both AMK blocks and connection blocks. By referring to Figure 5, we also observe that the first register bit is equivalent to the first variable, i.e. $R_{1,1,1} \equiv x_4$. Leveraging this insight, we can substitute that register bit with its corresponding variable to further decrease the number of auxiliary variables required by each block by one, resulting in $wk - \frac{k^2+k}{2} - 1$. Consequently, the total number of auxiliary variables needed for the entire Ladder constraint becomes $(2M-2)(wk - \frac{k^2+k}{2} - 1)$.

The analysis above demonstrates the efficiency of our proposed encoding. In terms of the required auxiliary variables and the number of generated clauses, the $SCL_{\leq k}$ encoding outperforms the *Seq*, *BDD*, and *Card* encodings. Specifically, while the latter three encodings require more than $\mathcal{O}(n^2)$ auxiliary variables and clause space, $SCL_{\leq k}$ only needs $\mathcal{O}(Mwk)$, which is approximately $\mathcal{O}(nk)$. Additionally, although the *Pairwise* encoding does not utilize auxiliary variables, its complexity of $\mathcal{O}(n^{k+1})$ is significantly higher than our linear complexity of $\mathcal{O}(nk)$. We provide a more comprehensive demonstration in the experimental evaluation (Section 6.1.2), which clarifies the comparative analysis and highlights the superiority of $SCL_{\leq k}$ over *Pairwise*, *Seq*, *BDD*, and *Card* encodings.

4 Encodings for Ladder AMO constraint

In this section, we introduce the *SCL* encoding formulas for the Ladder AMO constraint ($SCL_{\leq 1}$), derived from those for the Ladder AMK constraint ($SCL_{\leq k}$). Additionally, we present *Duplex* encoding [14], a decomposition-based encoding that has proven to be an effective encoding for the Ladder AMO constraint.

4.1 $SCL_{\leq 1}$ encoding

a. ***Encoding each block in the Ladder AMO constraint***

The Ladder AMO constraint is essentially a specific instance of the Ladder AMK constraint. Therefore, its block encoding can be directly derived from that of the Ladder AMK. When $k = 1$, the formulas from (9) to (13) below are utilized to encode each block in the Ladder AMO constraint.

$$\bigwedge_{j=1}^{w-1} x_{i,j} \rightarrow R_{i,j,1} \overset{k=1}{\Rightarrow} \bigwedge_{j=1}^{w-1} x_{i,j} \rightarrow R_{i,j,1} \tag{9}$$

$$\bigwedge_{j=2}^{w-1} \bigwedge_{s=1}^{\min(j-1,k)} R_{i,j-1,s} \rightarrow R_{i,j,s} \overset{k=1}{\Rightarrow} \bigwedge_{j=2}^{w-1} R_{i,j-1,1} \rightarrow R_{i,j,1} \tag{10}$$

$$\bigwedge_{j=2}^{w-1} \bigwedge_{s=2}^{\min(j,k)} x_{i,j} \wedge R_{i,j-1,s-1} \rightarrow R_{i,j,s} \overset{k=1}{\Rightarrow} \emptyset$$

$$\bigwedge_{j=1}^{k} \neg x_{i,j} \rightarrow \neg R_{i,j,j} \overset{k=1}{\Rightarrow} \neg x_{i,1} \rightarrow \neg R_{i,1,1} \tag{11}$$

$$\bigwedge_{j=2}^{w-1} \bigwedge_{s=2}^{min(j,k)} \neg R_{i,j-1,s-1} \rightarrow \neg R_{i,j,s} \overset{k=1}{\Rightarrow} \emptyset$$

$$\bigwedge_{j=2}^{w-1} \bigwedge_{s=1}^{min(j-1,k)} \neg x_{i,j} \wedge \neg R_{i,j-1,s} \rightarrow \neg R_{i,j,s} \overset{k=1}{\Rightarrow} \bigwedge_{j=2}^{w-1} \neg x_{i,j} \wedge \neg R_{i,j-1,1} \rightarrow \neg R_{i,j,1} \tag{12}$$

$$\bigwedge_{j=k+1}^{w} x_{i,j} \rightarrow \neg R_{i,j-1,k} \overset{k=1}{\Rightarrow} \bigwedge_{j=2}^{w} x_{i,j} \rightarrow \neg R_{i,j-1,1} \tag{13}$$

b. ***Clauses connecting two adjacent blocks***

Similar to the encoding of individual blocks, the connection between blocks in the Ladder AMO constraint can also be derived from the block connection in the Ladder AMK constraint (refer to formula 8) by setting $k = 1$. Specifically,

$$\bigwedge_{j=2}^{w}\bigwedge_{p=1}^{k}\Big(\sum_{q=1}^{w-j+1} x_{i,q} \leq k-p \vee \sum_{q'=1}^{j-1} x_{i+1,q'} \leq p-1\Big)$$

$$\Big\Downarrow k=1$$

$$\bigwedge_{j=2}^{w}\Big(\sum_{q=1}^{w-j+1} x_{i,q} \leq 0 \vee \sum_{q'=1}^{j-1} x_{i+1,q'} \leq 0\Big) \tag{14}$$

4.2 *Duplex* encoding

Fazekas et al. introduced the *Duplex* [14] for encoding Ladder AMO constraints. In this approach, the AMO and At-Most-Zero (AMZ) sub-expressions obtained through the decomposition of Ladder AMO constraints are encoded by applying *Binary Decision Diagrams* (*BDD*) encoding [21, 22] for corresponding windows of variables (note that the term "window" is equivalent to our term "subset"). Each window is encoded using two components: a forward *BDD* and a backward *BDD*. The forward *BDD* is constructed from the leftmost to the rightmost variable, while the backward *BDD* is constructed in the reverse direction, from right to left.

Figure 6 illustrates the construction of two *BDDs* with the variable arrangements of $x_1 \prec x_2 \prec x_3 \prec x_4$ and $x_8 \prec x_7 \prec x_6 \prec x_5$, respectively. This construction involves making decisions for each variable, using their values to branch the necessary conditions of the original constraints. For instance, to maintain the condition $x_1 + x_2 + x_3 + x_4 \leq 1$, if x_1 is *true*, then $x_2 + x_3 + x_4 \leq 0$ must hold; conversely, if x_1 is $false$, the condition $x_2 + x_3 + x_4 \leq 1$ must be satisfied.

To ensure there is at most one variable assigned *true* within the AMO constraints of Ladder AMO constraint, *Duplex* encoding bonds the constructed *BDDs* together. Specifically, for each successive pair of windows $\{\omega_i, \omega_{i+1}\}$, where ω_i^f is the forward *BDD* of ω_i and ω_{i+1}^b is the backward *BDD* of ω_{i+1}, the j^{th} layer of ω_i^f is connected to the $(w-j+2)^{th}$ layer of ω_{i+1}^b, where $2 \leq j \leq w$. For instance, bonding two *BDDs* as shown in Figure 6 requires the following clauses.

$$\begin{aligned}
&\wedge(x_2 + x_3 + x_4 \leq 1) \wedge (x_5 \leq 1) \wedge ((x_2 + x_3 + x_4 \leq 0) \vee (x_5 \leq 0))\\
&\wedge(x_3 + x_4 \leq 1) \wedge (x_5 + x_6 \leq 1) \wedge ((x_3 + x_4 \leq 0) \vee (x_5 + x_6 \leq 0))\\
&\wedge(x_4 \leq 1) \wedge (x_5 + x_6 + x_7 \leq 1) \wedge ((x_4 \leq 0) \vee (x_5 + x_6 + x_7 \leq 0))
\end{aligned}$$

Leveraging the benefits of reusing shared sub-expressions, the *Duplex* encoding achieves significantly enhanced complexity compared to non-decompose encodings. As shown in the work of Fazekas et al., the *Duplex* encoding has a complexity of $\mathcal{O}(n)$ [14], whereas all non-decompose encodings exhibit complexity of $\mathcal{O}(n^2)$ (refer to Table 2). For a Ladder AMO constraint involving n variables and considering $M = \lceil \frac{n}{w} \rceil$

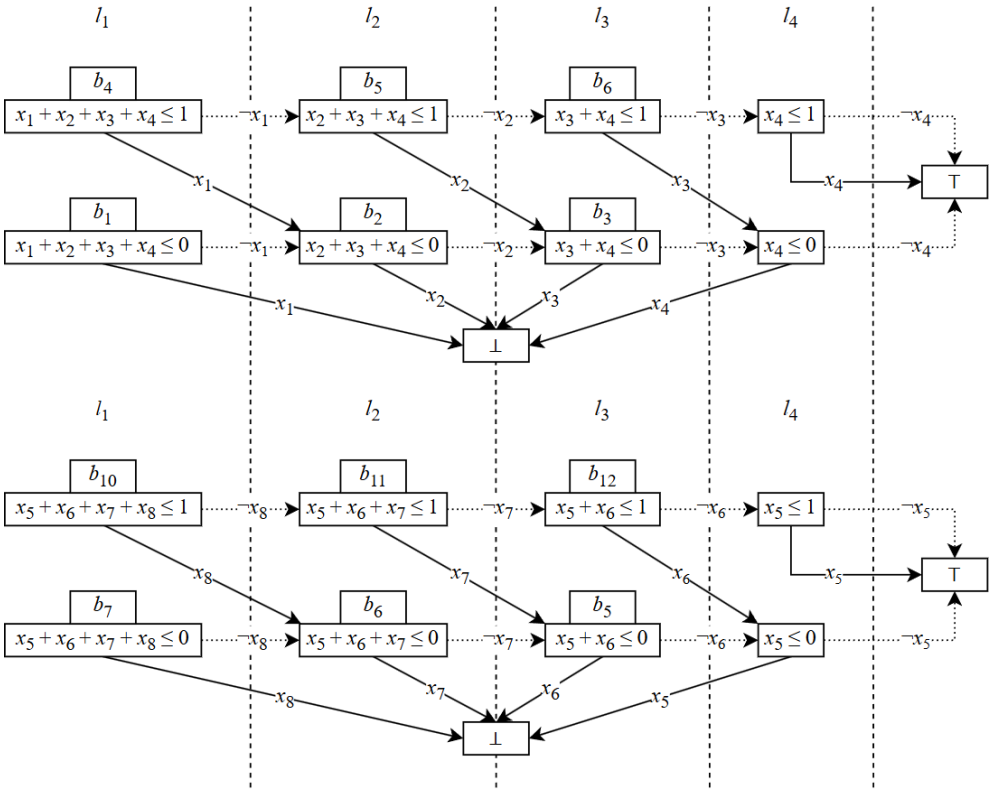

Fig. 6: Forward and backward *BDDs* for Ladder AMO constraint.

Source: Fazekas et al. [14]

windows of width w, with $N = (n-w)+1$ being the number of AMO constraints, this encoding generates around $13Mw - 14M - 3w + 2$ clauses and introduces additional $4M(w-1)$ auxiliary variables.

4.3 Encoding comparison for Ladder AMO constraints

In addition to the previously mentioned encodings, which are *Pairwise*, *Sequential Counter* (*Seq*), Binary Decision Diagrams (*BDD*), and *Cardinality Network* (*Card*), we compare $SCL_{\leq 1}$ with the *Duplex* encoding [14]. Moreover, among the group of encodings that individually handle each AMO constraint, we also include the *Product* encoding [23].

Table 2 presents a summary of the evaluation results for various encoding techniques used to encode Ladder AMO constraints across n variables with a width of w. The comparison is based on several criteria, including the number of required auxiliary variables needed, the total number of generated clauses, and the complexity of the clause space. This evaluation incorporates the study by Fazekas et al. [14] as well as our proposed $SCL_{\leq k}$ encoding for the case when $k = 1$.

Table 2: Size of SAT encoding for a Ladder AMO constraint over n variables with width w, $N = (n - w) + 1$ and $M = \lceil \frac{n}{w} \rceil$.

Encoding	Technique	Auxiliary variables	Clauses	Complexity
Pairwise	Encode each AMO constraint separately	0	$\frac{(w-1)w}{2} + (N-1)(w-1)$	$\mathcal{O}(n^2)$
Seq		$N(w-2)$	$N(4w-7)$	$\mathcal{O}(n^2)$
BDD		$2N(w-1)$	$4N(w-1)$	$\mathcal{O}(n^2)$
Card		$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Product		$N(2\sqrt{w} + \mathcal{O}\sqrt[4]{w})$	$N(2w + 4\sqrt{w} + \mathcal{O}\sqrt[4]{w})$	$\mathcal{O}(n^2)$
Duplex	Encode set of AMO constraints	$4M(w-1)$	$13Mw - 14M - 3w + 2$	$\mathcal{O}(n)$
$SCL_{\leq 1}$		$(2M-2)(w-2)$	$8Mw - 14M - 7w + 13$	$\mathcal{O}(n)$

To represent AMO constraints, the *Cardinality Network* (*Card*) encoding selects $k = 2$ and generates two outputs, i.e. c_1 and c_2. Then, the output c_2 is imposed to be *false*, ensuring that no more than one input variable can be assigned *true*. As a result, the number of auxiliary variables, required clauses, and the overall clause complexity of the *Card* encoding is $\mathcal{O}(n^2 \log_2^2 2) = \mathcal{O}(n^2)$.

The *Product* encoding is based on arranging variables in a two-dimensional coordinate system. The values along each axis are treated as additional variables and are then organized into a smaller two-dimensional coordinate system. This process is recursively applied until further decomposition into smaller coordinate systems is no longer possible. Through this representation technique, *Product* encoding utilizes $N(2\sqrt{w} + \mathcal{O}\sqrt[4]{w})$ auxiliary variables and $N(2w + 4\sqrt{w} + \mathcal{O}\sqrt[4]{w})$ clauses to encode a Ladder AMO constraint.

The *Duplex* encoding employs the decomposition technique and encodes each window separately by generating two *BDDs* with distinct variable arrangements. After that, these constructed windows are bonded together to maintain the AMO property of the original constraints. As presented in Section 4.2, the *Duplex* encoding requires approximately $13Mw - 14M - 3w + 2$ clauses and $4M(w-1)$ auxiliary variables.

The analysis presented above underscores the efficiency of our proposed encoding. Like the $SCL_{\leq k}$ encoding, the $SCL_{\leq 1}$ encoding outperforms several alternatives, including *Pairwise*, *Seq*, *BDD*, and *Card* encodings. As detailed in Table 2, the complexity of $SCL_{\leq 1}$ is $\mathcal{O}(n)$, which is also superior to that of the *Product* encoding, which has a complexity of $\mathcal{O}(n^2)$. Furthermore, among linear complexity encodings, $SCL_{\leq 1}$ utilizes roughly half the number of variables and clauses compared to the *Duplex* encoding. To distinctly showcase the outstanding performance of our encoding, we present two figures: Figures 7 and 8, which illustrate the performance efficiency of the $SCL_{\leq 1}$ encoding. Figure 7 depicts the number of clauses, while Figure 8 illustrates

the number of auxiliary variables required to encode a set of Ladder AMO constraints for $n = 1000$ and $w \in (1, 1000)$, based on the calculations presented in Table 2.

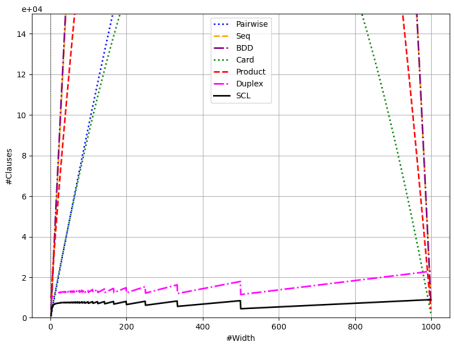

Fig. 7: Comparison of the number of clauses of a Ladder AMO constraint with parameters set to $n = 1000$ and $w \in (1, 1000)$.

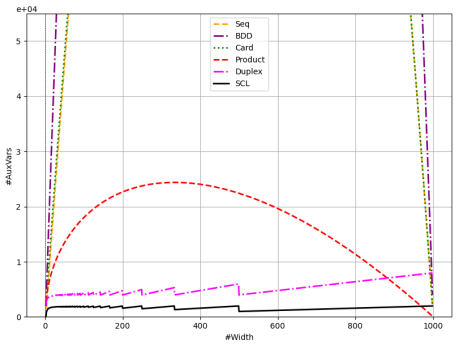

Fig. 8: Comparison of the number of auxiliary variables of a Ladder AMO constraint with parameters set to $n = 1000$ and $w \in (1, 1000)$.

5 SAT solving for the Anti-bandwidth problem

Anti-bandwidth problem (ABP) is an NP-hard problem with applications in various scheduling scenarios [24], including radio frequency assignment [25], obnoxious facility location [26] and map coloring [27]. In this section, we present SAT formulae including the Ladder AMO constraints for solving ABP.

5.1 Problem definition

Let $G = (V, E)$ be a graph where V and E denote the set of vertices and edges, respectively. A labeling function f is a bijection from V to $\{1, 2, \ldots, |V|\}$, assigning a unique label to each vertex in V. The antibandwidth $AB_f(i)$ of a vertex $i \in V$ under the labeling function f is defined as the minimum absolute difference between the label of i and the labels of its adjacent vertices:

$$AB_f(i) = min\{|f(i) - f(i')| : \{i, i'\} \in E\}$$

The bandwidth of G under the labeling function f, denoted as $AB_f(G)$, is the minimal value among the $AB_f(i)$ values:

$$AB_f(G) = min\{AB_f(i) : i \in V\}$$

The Anti-Bandwidth Problem aims to find a labeling function of a graph that maximizes its bandwidth. The maximum bandwidth value of a graph G is referred to as the anti-bandwidth ($AB(G)$) of the graph.

$$AB(G) = \max AB_f(G)$$

5.2 ILP model for ABP

The ABP is expressible as an integer linear programming (ILP) [10] model with binary (0–1) variables. By defining x_i^l to be a boolean variable that holds *true* if vertex i is assigned label l, and $false$ otherwise, this model employs two formulas, (VERTICES) and (LABELS), to guarantee that every vertex in the graph receives a unique label. Specifically, formula (VERTICES) ensures that each label is allocated to exactly one vertex, while formula (LABELS) guarantees that each vertex is given precisely one label. Combining these two equations preserves the one-to-one relationship between vertices and labels.

$$\sum_{i \in V} x_i^l = 1 \qquad \forall l \in \{1, \ldots, |V|\} \qquad \text{(VERTICES)}$$

$$\sum_{l=1}^{|V|} x_i^l = 1 \qquad \forall i \in V \qquad \text{(LABELS)}$$

Based on this approach, Markus Sinnl [10] proposed an iterative formulation, framed as a feasibility problem, to answer the question: "Does there exist a solution with $AB(G) \geq k+1$?". This formulation includes both formulas (VERTICES) and (LABELS), along with formula (OBJ-k), which ensures that the difference between the labels of adjacent vertices i and i' is strictly greater than k whenever $k+1$ is a feasible bandwidth.

$$\sum_{l'=l_2}^{l_{2+k}} (x_i^{l'} + x_{i'}^{l'}) \leq 1 \qquad \forall \{i, i'\} \in E, \forall 1 \leq l_2 \leq |V| - k \qquad \text{(OBJ-k)}$$

$$\begin{aligned}
x_i^1 + x_i^2 + x_i^3 + x_i^4 + x_{i'}^1 + x_{i'}^2 + x_{i'}^3 + x_{i'}^4 &\leq 1 \\
x_i^2 + x_i^3 + x_i^4 + x_i^5 + x_{i'}^2 + x_{i'}^3 + x_{i'}^4 + x_{i'}^5 &\leq 1 \\
x_i^3 + x_i^4 + x_i^5 + x_i^6 + x_{i'}^3 + x_{i'}^4 + x_{i'}^5 + x_{i'}^6 &\leq 1 \\
x_i^4 + x_i^5 + x_i^6 + x_i^7 + x_{i'}^4 + x_{i'}^5 + x_{i'}^6 + x_{i'}^7 &\leq 1 \\
x_i^5 + x_i^6 + x_i^7 + x_i^8 + x_{i'}^5 + x_{i'}^6 + x_{i'}^7 + x_{i'}^8 &\leq 1 \\
x_i^6 + x_i^7 + x_i^8 + x_i^9 + x_{i'}^6 + x_{i'}^7 + x_{i'}^8 + x_{i'}^9 &\leq 1 \\
x_i^7 + x_i^8 + x_i^9 + x_i^{10} + x_{i'}^7 + x_{i'}^8 + x_{i'}^9 + x_{i'}^{10} &\leq 1
\end{aligned}$$

Fig. 9: OBJ-k constraint of $\{i, i'\}$ with $|V| = 10$ and $k = 3$.

For the ABP problem with 10 vertices ($|V| = 10$) and $k = 3$, the constraints given by equation (OBJ-k) for any pair of vertices i and i' are depicted in Figure 9. These constraints ensure that the labels l_i and $l_{i'}$ of vertices i and i', respectively, always satisfy the condition $|l_i - l_{i'}| \leq k + 1 = 4$. As an example, if $l_i = 4$, meaning x_i^4 holds *true*, then to prevent the set of AMO constraints (in Figure 9) from being violated, $x_{i'}^1, x_{i'}^2, x_{i'}^3, x_{i'}^4, x_{i'}^5, x_{i'}^6$, and $x_{i'}^7$ must all be *false*, which implies that $l_{i'} \notin \{1, 2, 3, 4, 5, 6, 7\}$. The remaining possible labels for vertex i' are $\{8, 9, 10\}$, ensuring that the label distance condition between vertices is maintained.

5.3 SAT implementation of ABP constraints

5.3.1 Decomposition of $OBJ - k$ constraints

Encoding the equation (OBJ-k) requires a total of $|E|$ separate constraints, each corresponding to an edge in the set E. However, in most graphs, the number of edges is significantly greater than the number of vertices. To optimize the encoding of (OBJ-k), Fazekas et al. [14] employ Lemma 1 to decompose the constraints of (OBJ-k) as follows:

$$\sum_{l'=l_2}^{l_{2+k}} (x_i^{l'} + x_{i'}^{l'}) \leq 1$$

$$\overset{\text{Lemma 1}}{\equiv}$$

$$\sum_{l'=l_2}^{l_{2+k}} x_i^{l'} \leq 1 \wedge \sum_{l'=l_2}^{l_{2+k}} x_{i'}^{l'} \leq 1 \wedge (\sum_{l'=l_2}^{l_{2+k}} x_i^{l'} \leq 0 \vee \sum_{l'=l_2}^{l_{2+k}} x_{i'}^{l'} \leq 0)$$

By using this decomposition technique, the constraint from Equation (OBJ-k) for any edge $\{x_i, x_{i'}\}$ can be represented through the corresponding vertex-level constraints of x_i and $x_{i'}$. These vertex-level constraints can be reused to represent multiple edges, which reduces the total number of constraints needed from $|E|$ to $|V|$.

Applying Lemma 1 to decompose the constraint from Equation (OBJ-k) as shown in Figure 9, we derive two new constraints, illustrated in Figure 10. These constraints can be encoded using the $SCL_{\leq 1}$ encoding discussed in Section 3.

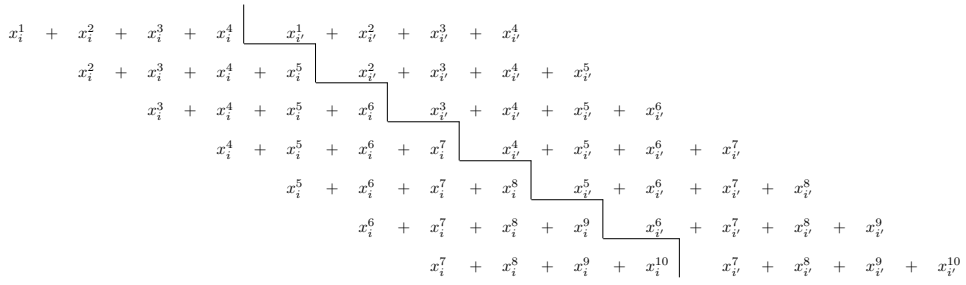

Fig. 10: Decomposition of OBJ-k constraint with $|V| = 10$ and $k = 3$

Additionally, this decomposition uses a disjunction to connect the constraints outlined in (OBJ-k) across various vertices, thereby ensuring that the AMO property is

maintained for the (OBJ-k) constraint over edges. The disjunctions can be formulated by combining AMZ sub-expressions within the $SCL_{\leq 1}$ encoding of the corresponding (OBJ-k) constraints, in accordance with Proposition 2.

Proposition 2. *A constraint* $x_1 + x_2 + \ldots + x_n \leq 0$ *holds iff for all* $1 \leq i < n$ *:*
$$(x_1 + \ldots + x_i \leq 0) \wedge (x_{i+1} + \ldots + x_n \leq 0)$$

For instance, the decomposition of $x_i^2+x_i^3+x_i^4+x_i^5+x_{i'}^2+x_{i'}^3+x_{i'}^4+x_{i'}^5 \leq 1$ utilizes *disjunction* $(x_i^2+x_i^3+x_i^4+x_i^5 \leq 0) \vee (x_{i'}^2+x_{i'}^3+x_{i'}^4+x_{i'}^5 \leq 0)$ (see Figure 10). Applying Proposition 2, the constraints $x_i^2+x_i^3+x_i^4+x_i^5 \leq 0$ and $x_{i'}^2+x_{i'}^3+x_{i'}^4+x_{i'}^5 \leq 0$ can be further decomposed into $(x_i^2+x_i^3+x_i^4 \leq 0) \wedge (x_i^5 \leq 0)$ and $(x_{i'}^2+x_{i'}^3+x_{i'}^4 \leq 0) \wedge (x_{i'}^5 \leq 0)$, respectively. These clauses can then be substituted with the corresponding *Seq* register bits, based on the block decomposition illustrated in Figure 2.

To enhance our approach, we implement the constraint (LABELS) by utilizing auxiliary variables when constructing $SCL_{\leq 1}$ for each vertex in the graph. This strategy improves the optimization of the generated clauses, providing a more efficient alternative to encoding with x_i^l for the constraint (LABELS). The remaining (VERTICES) constraint is applied exactly one encoding in SAT using both *Product* and *Seq* encodings.

5.3.2 Symmetry breaking

Referring to Markus Sinnl [10], for every labeling f of an n-vertex graph, there exists a reversed labeling f', where each label in f' is computed from its corresponding label in f using the following linear transformation:

$$f' = n + 1 - f.$$

This transformation ensures a one-to-one correspondence between the values in f and those in f'. Consequently, if f satisfies $(VERTICES)$ and $(LABELS)$ constraints, then f' also satisfies them. Moreover, this transformation also ensures that f' maintains the same bandwidth as f, which is proven as follows.

$$\begin{aligned} |f'(i) - f'(i')| &= |(n+1-f(i)) - (n+1-f(i'))| \\ &= |f(i') - f(i)| \end{aligned}$$

Based on this observation, we apply the symmetry breaking technique [28] to reduce the search space. In our implementation of ABP, we employ symmetry breaking at one selected node using two different configurations: the first node and the highest-degree node.

6 Experimental Evaluation

We conducted two experiments to evaluate the effectiveness of our proposed SAT encoding SCL. The first experiment compares the solving times of SCL against other SAT encodings when addressing single Ladder AMO constraints ($k = 1$) and Ladder AMK constraints ($k > 1$). The second experiment focuses on evaluating $SCL_{\leq 1}$ in solving the Anti-bandwidth Problem. These experiments were performed on a cluster

provided by the *Google Cloud Platform*[1] featuring *8 vCPUs, 4 cores, and 64GB of memory*, running the *Debian GNU/Linux 12 operating system*. The SAT solver used in the experiments was CaDiCaL version 1.2.1 [29].

6.1 Single Ladder AMK constraint

The number of variables and clauses generated by the $SCL_{\leq 1}$ and $SCL_{\leq k}$ encodings, in comparison to other SAT encodings, is detailed in Tables 1 and 2 of the paper. In this experiment, we aim to evaluate the solving time of SCL relative to other SAT encodings when given a Ladder AMK constraint as input, while varying the number of variables and the width w.

6.1.1 Solving time of the Ladder AMO constraint

The experimental dataset for the Ladder AMO constraint consists of a fixed number of variables, specifically $n = 1000$. The width w ranges from 5 to 995 in increments of 5 (i.e., $5, 10, 15, ..., 995$). The different SAT encodings compared to the $SCL_{\leq 1}$ are detailed in Table 2. These encodings include *Pairwise*, *Sequential Counter (Seq)*, *Binary Decision Diagrams* (*BDD*), *Cardinality Network* (*Card*), *Product*, and *Duplex*. Figures 11 and 12 illustrate the solving times for the various encodings.

Among the top three encodings, as shown in Figure 12, $SCL_{\leq 1}$ emerged as the most effective, surpassing *Duplex*, which was previously recognized as an efficient encoding [14]. Overall, the results indicate that, in most cases, the $SCL_{\leq 1}$ encoding significantly outperforms all other SAT encodings in terms of time efficiency. Additionally, the effectiveness of the generated clauses and the necessary auxiliary variables for $SCL_{\leq 1}$ are discussed earlier in Section 4.3.

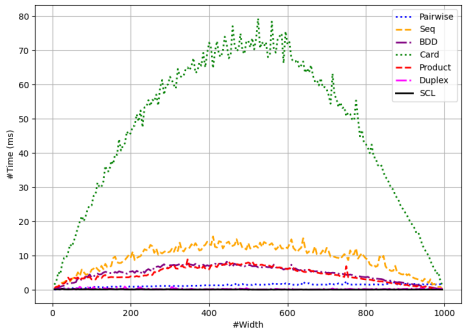

Fig. 11: Solving time of various encodings on the Ladder AMO constraints with $n = 1000$ and $w \in \{5, 10, 15, ..., 995\}$.

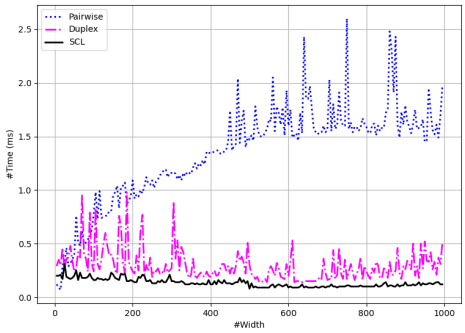

Fig. 12: Solving time of the top three encodings on the Ladder AMO constraints with $n = 1000$ and $w \in \{5, 10, 15, ..., 995\}$.

[1] https://console.cloud.google.com/compute

6.1.2 Solving time of the Ladder AMK constraint

To compare the solving times of $SCL_{\leq k}$ with other SAT encodings, we employed a dataset of Ladder AMK constraints that included n variables and a width w. The values for n were chosen from the range $\{100, 200, 300, \ldots, 5000\}$, while the values for w were taken from the set $\{10, 20, 50, 100\}$. We defined k as $k = \frac{1}{5}w$ (which means $k \in \{2, 4, 10, 20\}$). Each instance was allowed to run for a maximum of 600 seconds. The SAT encodings compared to $SCL_{\leq k}$ are detailed in Table 1, which includes *Pairwise*, *Seq*, *BDD*, and *Card*. The *Product* encoding is excluded from this experiment, as it is less efficient for solving AMK constraints.

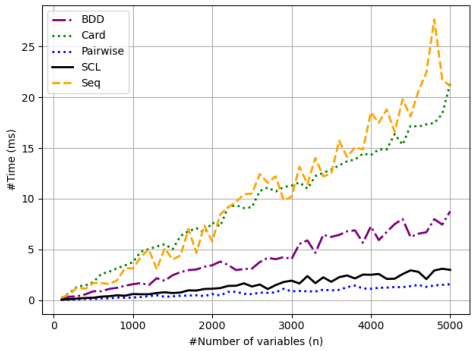

Fig. 13: Solving time of Ladder AMK constraints with $w = 10$ and $k = 2$.

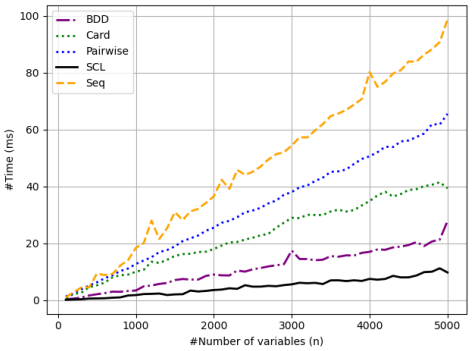

Fig. 14: Solving time of Ladder AMK constraints with $w = 20$ and $k = 4$.

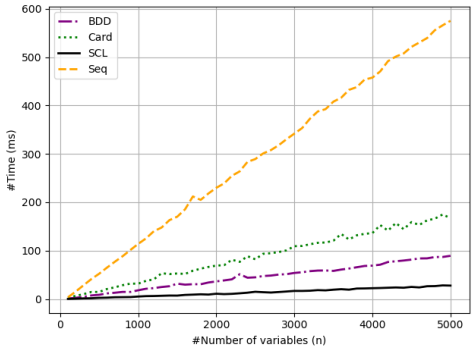

Fig. 15: Solving time of Ladder AMK constraints[2] with $w = 50$ and $k = 10$.

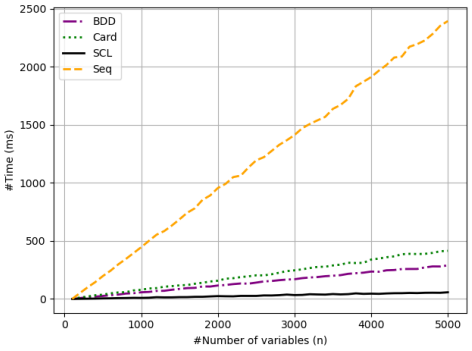

Fig. 16: Solving time of Ladder AMK constraints[2] with $w = 100$ and $k = 20$.

[2] *Pairwise* encoding times out for all input sizes n within the 600-second limit.

The solving times of five different SAT encodings for four values of k are illustrated in Figures 13, 14, 15, and 16. The encoding $SCL_{\leq k}$ achieves the fastest solving times for $k = 4, 10$, and 20, and it ranks second after the *Pairwise* encoding for $k = 2$. In comparison to *Seq*, *BDD*, and *Card*, $SCL_{\leq k}$ consistently demonstrates superior performance regarding solving time.

For instance, when $n = 5000$, the solving time for $SCL_{\leq k}$ is approximately one-third of the time required by the second fastest encoding, *BDD*. Specifically, for $k = 4$, $SCL_{\leq k}$ solves the problem in 9.7 milliseconds, while *BDD* takes 27.8 milliseconds. For $k = 10$, $SCL_{\leq k}$ takes 27.7 milliseconds compared to *BDD*'s 89.2 milliseconds. For $k = 20$, $SCL_{\leq k}$ finishes in 57.1 milliseconds, while *BDD* takes 288.8 milliseconds. The *Pairwise* encoding is the fastest for $k = 2$ but begins to slow down at $k = 4$. It times out for all input sizes n within the 600-second limit when solving for $k = 10$ and $k = 20$ (as shown in Figures 15 and 16).

6.2 Computation Results for the Anti-Bandwidth Problem

6.2.1 Datasets and Implementation Details

The second framework evaluates the effectiveness of the proposed encoding in addressing the Anti-Bandwidth Problem. It utilizes a dataset consisting of 24 matrices from the Harwell-Boeing Sparse Matrix Collection [30]. This collection includes 12 small instances and 12 large instances. Detailed information about these 24 matrices can be found in Table 3, which lists the names of the matrices, the number of vertices $|V|$, and the number of edges $|E|$. Additionally, the corresponding lower and upper bounds (LB and UB) are provided by Fazekas et al. [14].

Table 3: Harwell-Boeing Sparse Matrix benchmark.

Small instances

Instance	|V|	|E|	LB	UB
A-pores_1	30	103	6	8
B-ibm32	32	90	9	9
C-bcspwr01	39	46	16	17
D-bcsstk01	48	176	8	9
E-bcspwr02	49	59	21	22
F-curtis54	54	124	12	13
G-will57	57	127	12	14
H-impcol_b	59	281	8	8
I-ash85	85	219	19	27
J-nos4	100	247	32	40
K-dwt__234	117	162	46	58
L-bcspwr03	118	179	39	39

Large instances

Instance	|V|	|E|	LB	UB
M-bcsstk06	420	3720	28	72
N-bcsstk07	420	3720	28	72
O-impcol_d	425	1267	91	173
P-can__445	445	1682	78	120
Q-494_bus	494	586	219	246
R-dwt__503	503	2762	46	71
S-sherman4	546	1341	256	272
T-dwt__592	592	2256	103	150
U-662_bus	662	906	219	220
V-nos6	675	1290	326	337
W-685_bus	685	1282	136	136
X-can__715	715	2975	112	142

In this framework, we implemented the $SCL_{\leq 1}$ encoding alongside four SAT-based encodings: *Product*, *Pairwise*, *Sequential Counter*, and *Duplex*. Throughout the implementation, we followed a consistent sequential process. We began by setting the lower bound (LB) as the starting width for the Ladder AMO constraints. If the SAT solver returns a satisfiable (SAT) result, we increase the width by 1 and restart the process. Conversely, if the solver returns unsatisfiable (UNSAT) or reaches the upper bound (UB), it indicates that we have found the optimal solution, and the iteration process concludes.

In addition to the sequential implementation (SCL_{seq}), which incrementally solves for bandwidth values from LB to UB, we also implemented a multi-process approach for the proposed encoding SCL, referred to as SCL_{par}. This implementation initializes multiple processes, with each process invoking a SAT solver to address a different bandwidth. The multi-process approach is more efficient than the sequential method because if one process finds a solution for a higher bandwidth, it can terminate other processes working on smaller bandwidths. This allows the search to concentrate on higher bandwidths. In this experiment, SCL_{par} is executed with up to four parallel threads invoking the SAT solver.

Constraint Programming (CP) and Mixed Integer Programming (MIP) are included in this experiment, which are $F_e(k)$, *CPLEX*, *CP-MZ-Chuffed*, *CP-SAT*, and *Gurobi*. In our experiment, the $F_e(k)$ encoding leverages the MIP APIs from IBM ILOG CPLEX Optimization Studio[3] version 20.1, a robust framework that integrates constraint propagation with advanced optimization techniques for efficient problem-solving. The *CPLEX* approach utilizes the CP APIs of IBM ILOG CPLEX Optimization Studio version 22.1.1. The *CP-MZ-Chuffed* method employs the MiniZinc language [31] version 2.8.6 and incorporates Chuffed solver[4] version 0.13.2, which excels in lazy clause generation and is particularly effective for combinatorial problems. The *CP-SAT* approach relies on the CP-SAT Solver, a versatile solver designed for constraint satisfaction and optimization, leveraging multi-threading and SAT-based techniques to handle complex constraints, which is provided by Google OR-Tools[5] version 9.11.4210. Finally, the *Gurobi* encoding harnesses the Gurobi Optimizer[6] version 11.0.3, a powerful tool for mathematical optimization, known for its efficiency in handling large-scale constraint-based problems. Note that *CPLEX*, *Gurobi*, and *CP-SAT* are powerful CP and MIP solvers that have been under development for a long time.

6.2.2 Performance evaluation

Table 4 summarizes our experimental results on 24 selected matrices, with a time limit of 1800 seconds and a memory limit of 30 GB applied to the *Product*, *Pairwise*, *Sequential Counter* (Seq), *Duplex*, SCL_{seq}, SCL_{par}, $F_e(k)$, *CPLEX*, *CP-MZ-Chuffed*, *CP-SAT*, and *Gurobi* approaches. For each problem, the table displays the best solution found by each method, along with the solving time (in seconds) and memory consumption (in MB). The best anti-bandwidth value is highlighted in bold, and the best solving time is underlined. If the computation time exceeds 1800 seconds or the

[3] https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer
[4] https://github.com/chuffed/chuffed.git
[5] https://developers.google.com/optimization/cp/cp_solver
[6] https://www.gurobi.com/

Table 4: ABP solving results of different approaches with TO = 1800s and MO = 30GB.

Instance	LB	UB	*Product*			*Pairwise*			Seq			*Duplex*			SCL_{seq}			SCL_{par}			$F_c(k)$			*CPLEX*			*CP-MZ-Chuffed*			*CP-SAT*			*Gurobi*		
			k	Time(s)	MB	k	Time(s)	MB	k	Time(s)	MB	Obj-k	Time(s)	MB	Obj-k	Time(s)	MB	Obj-k	Time(s)	MB	Obj-k	Time(s)	MB	Obj-k	Time(s)	MB	Obj-k	Time(s)	MB	Obj-k	Time(s)	MB	Obj-k	Time(s)	MB
A-pores_1	6	8	**6**	122	77.5	**6**	65.7	56.9	**6**	47.7	55.4	**6**	3.4	14.1	**6**	9.15	19.8	**6**	8.8	23.6	**6**	3.53	74.9	7	TO	579.3	**6**	4.75	42.3	**6**	5.17	144.9	**6**	9.84	84.9
B-ibm32	9	9	**9**	6.02	48.9	**9**	18.7	34.5	**9**	2.13	42.9	**9**	0.19	8.8	**9**	0.3	7.8	**9**	0.21	9.2	**9**	4.57	106.8	**9**	0.39	41.3	**9**	1.06	29.3	**9**	1.38	120.5	**9**	6.99	108.2
C-bcspwr01	16	17	**17**	5.25	41.4	**17**	13	40.1	**17**	20.7	58.2	**17**	1.11	11.2	**17**	4.19	18.6	**17**	3.44	16.8	**17**	3.59	97.5	**17**	0.67	41.0	**17**	0.43	28.6	**17**	1.46	126.1	**17**	5.1	104.6
D-bcsstk01	8	9	**9**	9.43	140.1	**9**	5.36	48.2	**9**	11.0	118.9	**9**	0.54	14.9	**9**	0.19	12.6	**9**	0.13	13.2	**9**	14.9	147.0	**9**	2.28	46.8	**9**	1.12	41.0	**9**	1.72	158.4	**9**	15.9	252.9
E-bcspwr02	21	22	**21**	8.79	83.5	**21**	9.22	45.8	**21**	9.91	86.1	**21**	2.15	14.9	**21**	1.52	11	**21**	0.74	19.3	**21**	17.8	170.2	**21**	15.83	51.5	**21**	0.8	31.5	**21**	3.83	172.6	**21**	14.3	198.5
F-curtis54	12	13	**13**	13.6	141.6	**13**	1.52	41.3	**13**	11.1	127	**13**	0.27	14.1	**13**	0.28	12.3	**13**	0.13	23.9	**13**	13.3	145.8	**13**	2.5	47.2	**13**	1.35	38.8	**13**	2.6	179.8	**13**	14.3	255
G-will57	12	14	**13**	16.6	147.9	**13**	11.9	73.2	**13**	23.2	133.8	**13**	0.29	14.4	**13**	0.47	13.9	**13**	0.44	26.5	**13**	18.5	175.6	**13**	14.18	51.5	**13**	1.09	39.9	**13**	2.16	176.2	**13**	21.5	271.3
H-impcol_b	8	8	**8**	4.38	177	**8**	0.38	52.8	**8**	5.22	147.7	**8**	0.26	21.0	**8**	0.23	16.8	**8**	0.08	4.8	**8**	3.12	167.4	**8**	0.32	44.7	**8**	0.86	55.5	**8**	2.06	190.2	**8**	8.17	262.5
I-ash85	19	27	22	TO	870.8	22	TO	843.6	23	TO	619.8	**24**	TO	272	**24**	TO	381	**24**	TO	496.6	22	TO	460.3	23	TO	288.2	23	TO	374.8	**24**	TO	1723	22	TO	1219
J-nos4	32	40	33	TO	1119.5	33	TO	1070.1	-	TO	1572.8	**35**	472	177	**35**	980	503	**35**	720.3	505.5	-	TO	927.0	35	TO	272.7	**35**	384.4	343.9	36	TO	1818	33	TO	1690
K-dwt_234	46	58	49	TO	1373.3	51	TO	1192.6	48	TO	1110	51	TO	685	51	TO	530	**52**	TO	1377	50	TO	895.3	**52**	TO	243.8	47	TO	244.6	51	TO	1088	50	TO	784.4
L-bcspwr03	39	39	**39**	15.1	682.1	**39**	3.76	436.5	**39**	47.7	856.6	**39**	0.58	53	**39**	0.69	40.9	**39**	0.65	41	**39**	11.15	559.0	**39**	0.25	48.2	**39**	2.34	66.6	**39**	7.15	482.4	**39**	65.9	321.3
M-bcsstk06	28	72	-	-	MO	31	TO	22469	-	-	MO	**35**	TO	1726	**35**	TO	1398	**35**	TO	5076	-	TO	12207	30	TO	738.4	-	TO	3016	33	TO	5243	-	320.3	MO
N-bcsstk07	28	72	-	-	MO	31	TO	22470	-	-	MO	**35**	TO	1726	**35**	TO	1397	**35**	TO	5291	-	TO	12244	30	TO	737.9	-	TO	3016	33	TO	5250	-	304.1	MO
O-impcol_d	91	173	-	-	MO	96	TO	22681	-	-	MO	101	TO	1745	102	TO	1802	106	TO	5366	-	-	MO	**121**	TO	491.3	-	TO	1410	93	TO	5388	-	341	MO
P-can_445	78	120	-	-	MO	-	TO	27109	-	-	MO	-	TO	2464	79	TO	2156	**81**	TO	9758	-	-	MO	79	TO	869.2	-	TO	1843	-	TO	5732	-	411.6	MO
Q-494_bus	219	246	-	TO	25945	-	TO	29391	-	-	MO	-	TO	1443	**220**	TO	1301	**220**	TO	4713	-	TO	23775	**220**	TO	701.6	-	TO	763.8	-	316.8	MO	-	TO	18775
R-dwt_503	46	71	-	-	MO	-	-	MO	-	-	MO	**64**	TO	1634	**64**	TO	1474	**64**	TO	4991	-	TO	10633	56	TO	717.6	-	TO	1950	49	TO	7347	-	424.5	MO
S-sherman4	256	272	-	-	MO	-	-	MO	-	-	MO	**-**	TO	1326	**-**	TO	1073	**-**	TO	4003	-	-	MO	-	TO	689.9	-	TO	888.8	-	TO	29192	-	1013	MO
T-dwt_592	103	150	-	-	MO	-	-	MO	-	-	MO	-	TO	5357	**104**	TO	2619	**104**	TO	11445	-	-	MO	**104**	TO	1366	-	TO	1905	-	TO	9331	-	871.3	MO
U-662_bus	219	220	-	-	MO	-	-	MO	-	-	MO	**220**	471	2498	**220**	144	1377	**220**	122.9	2600	-	-	MO	**220**	3.15	250.4	-	TO	1061	-	328	MO	-	851.3	MO
V-nos6	326	337	-	-	MO	-	-	MO	-	-	MO	-	TO	1836	-	TO	1532	-	TO	6317	-	-	MO	-	TO	967.9	-	TO	1168	-	1260	MO	-	1071	MO
W-685_bus	136	136	-	-	MO	-	-	MO	-	-	MO	**136**	10.7	1431	**136**	7.57	1164	**136**	7.6	1162	-	-	MO	**136**	1.95	332.6	-	TO	1325	**136**	73.3	8223	-	855	MO
X-can_715	112	142	-	-	MO	-	-	MO	-	-	MO	-	TO	3060	113	TO	4599	**116**	TO	10388	-	-	MO	113	TO	1380	-	TO	2442	-	TO	11514	-	1013	MO

memory usage surpasses 30 GB, a termination signal is triggered, resulting in the process stopping due to a timeout (TO) or memory overload (MO), respectively. In such cases, the highest solved bandwidth is reported as the best result achieved by the technique. If the method fails to solve the problem using the initial lower bound (LB) value, the result is indicated with a "-" character.

Figures 17 and 18 compare the number of generated clauses and auxiliary variables used in the ABP encoding across 24 instances. This evaluation considers both the *SCL* and *Duplex* encodings, with *Duplex* recognized as the best encoding according to [14]. Each instance is represented by the first letter of its name (e.g., *A* for *A-pores_1* and *X* for *X-can__715)*. We analyzed the encodings at the maximum bandwidth achievable by both encodings within specified time and memory limits. The results show that as the instance size grows, indicated by the increasing number of vertices $|V|$ and edges $|E|$, the *SCL* encoding requires fewer clauses and auxiliary variables than the *Duplex* encoding. This finding highlights why *SCL* can outperform *Duplex* in solving certain ABP instances, particularly in 12 large instances, and it also demonstrates more efficient memory usage.

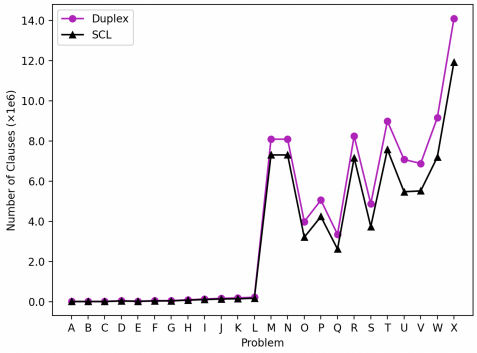

Fig. 17: Comparison of generated clauses between *SCL* and *Duplex*.

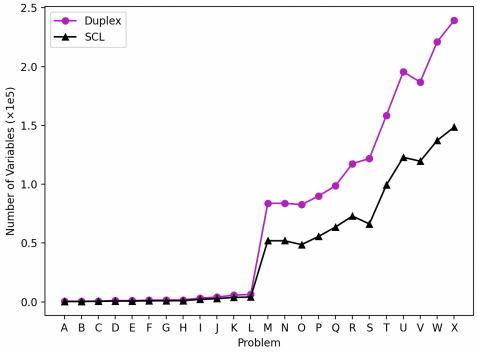

Fig. 18: Comparison of auxiliary variables between *SCL* and *Duplex*.

When solving 12 small graph instances, the *SCL* encoding yields the best results compared to other methods. The performance of SCL_{seq} and *Duplex* is quite competitive; however, SCL_{par}, an improved version of SCL_{seq}, outperforms *Duplex* by achieving a maximum bandwidth of 51 on the instance *K-dwt__234*, while *Duplex* only reaches a bandwidth of 50. The two best-performing methods on this instance are SCL_{par} and *CPLEX*. When compared to *CPLEX*, which is one of the top-performing methods among CP and MIP approaches, SCL_{par} finds the optimal bandwidth values for the two instances *A-pores_1* and *J-nos4*, which are 6 and 35, respectively. In contrast, *CPLEX* only proves feasible bandwidths of 6 and 34 for those instances. Moreover, SCL_{par} also performs better on the *I-ash85* instance. Compared to the other efficient CP method, *CP-SAT*, SCL_{par} delivers better results on the instances *J-nos4* and *K-dwt__234*, while *CP-SAT* performs relatively weakly across the 12 large instances. MIP-based methods such as *Gurobi*, $F_e(k)$, and the remaining SAT encodings exhibit weaker performance than SCL_{par}.

Table 5 summarizes the experimental results on 12 large problem instances for the five best-performing approaches: *Duplex*, SCL_{seq}, SCL_{par}, *CPLEX*, and *CP-SAT*. Overall, SCL_{par} consistently delivers the best performance compared to other methods, except for the *O-impcol_d* instance, where *CPLEX* achieves significantly superior results. When compared with *CPLEX*, SCL_{par} outperforms it on five instances: *M-bcsstk06*, *N-bcsstk07*, *P-can__445*, *R-dwt__503*, and *X-can__715*. In terms of performance against *Duplex*, SCL_{par} shows clear superiority on the instances *O-impcol_d*, *P-can__445*, *Q-494_bus*, *T-dwt__592*, and *X-can__715*. Notably, on the

instances *P-can__445* and *X-can__715*, no other method produces better results than SCL_{par}.

Table 5: ABP solving results of the five best approaches in 12 large instances with TO = 1800s and MO = 30GB.

Instance	LB	UB	*Duplex*			SCL_{seq}			SCL_{par}			*CPLEX*			*CP-SAT*		
			Obj-k	Time(s)	MB	Obj-k	Time(s)	MB	Obj-k	Time(s)	MB	Obj-k	Time(s)	MB	Obj-k	Time(s)	MB
M-bcsstk06	28	72	**35**	TO	1726	**35**	TO	1398	**35**	TO	5076	30	TO	738.4	33	TO	5243
N-bcsstk07	28	72	**35**	TO	1726	**35**	TO	1397	**35**	TO	5291	30	TO	737.9	33	TO	5250
O-impcol_d	91	173	101	TO	1745	102	TO	1802	106	TO	5366	**121**	TO	491.3	93	TO	5388
P-can__445	78	120	-	TO	2464	79	TO	2156	**81**	TO	9758	79	TO	869.2	-	TO	5732
Q-494_bus	219	246	-	TO	1443	**220**	TO	1301	**220**	TO	4713	**220**	TO	701.6	-	316.8	MO
R-dwt__503	46	71	**64**	TO	1634	**64**	TO	1474	**64**	TO	4991	56	TO	717.6	49	TO	7347
S-sherman4	256	272	**-**	TO	1326	**-**	TO	1073	**-**	TO	4003	-	TO	689.9	-	TO	29192
T-dwt__592	103	150	-	TO	5357	**104**	TO	2619	**104**	TO	11445	**104**	TO	1366	-	TO	9331
U-662_bus	219	220	**220**	471	2498	**220**	144	1377	**220**	122.9	2600	**220**	3.15	250.4	-	328	MO
V-nos6	326	337	-	TO	1836	-	TO	1532	-	TO	6317	-	TO	967.9	-	1260	MO
W-685_bus	136	136	**136**	10.7	1431	**136**	7.57	1164	**136**	7.6	1162	**136**	1.95	332.6	**136**	73.3	8223
X-can__715	112	142	-	TO	3060	113	TO	4599	**116**	TO	10388	113	TO	1380	-	TO	11514

6.3 Discussion on the potential applicability of the newly proposed encoding $SCL_{\leq k}$

The experimental results presented above provide evidence for the effectiveness of the newly proposed encoding $SCL_{\leq k}$, as discussed in this paper. When applied to a single Ladder AMK constraint, $SCL_{\leq k}$ significantly outperforms existing encoding representations. Furthermore, when addressing a real-world problem such as ABP, the SCL encoding for AMO constraints continues to yield efficient solutions, performing competitively alongside established CP and MIP solvers like *CPLEX*, *CP-SAT*, and *Gurobi*.

However, our experiments have not yet provided clear evidence of an advantage for $SCL_{\leq k}$ when $k > 1$ in any specific combinatorial or scheduling problems that involve Ladder AMK constraints with $k > 1$. In the Car Sequencing Problem [1], the Ladder AMK constraints have a relatively small width (i.e., width $w = 5$). In cases with small widths (i.e., $w \leq 7$), $SCL_{\leq k}$ typically does not outperform direct encodings, such as the *Pairwise* encoding. In the context of sequence constraints found in the Employee Scheduling Problem [7], these constraints represent only a subset of the overall problem's constraints and include both At-Least-K and At-Most-K constraints. Therefore, the sequence constraints in this scheduling context emphasize the need for the SCL encoding to be not only developed for AMK but also extended to represent At-Least-K and AMONG constraints (i.e., $SCL_{\geq k}$ and $SCL_{[l,u]}$), based on the $SCL_{\leq k}$ formulation introduced in this paper.

7 Conclusion

SAT solving has emerged as a competitive and effective alternative to CP and MIP solvers for various combinatorial, scheduling, and OR problems. Alongside significant advancements in the capabilities of SAT solvers, the development of efficient encodings tailored to specific types of constraints can greatly enhance overall solution performance. Numerous studies have shown that optimizing SAT encodings can lead to substantial improvements, especially when replacing direct encoding techniques with more compact representations. These optimized encodings can reduce the number of auxiliary variables, generated clauses, and clause sizes.

This paper introduces a novel SAT encoding for sets of AMK cardinality constraints arranged in a ladder shape. Rather than encoding each AMK constraint individually, our approach generates encodings for blocks that represent adjacent AMK constraints with the aim of encoding a set of AMK constraints simultaneously. To our knowledge, previous research in this area has mainly focused on using *BDD* to represent ladder AMO constraints [14], and no proposed encodings exist for AMK constraints where $k > 1$. Compared to the *BDD*-based representation for AMO constraints, our encoding, which utilizes *Sequential Counter* encoding, shows significant improvements for AMO constraints and also demonstrates effectiveness for AMK constraints. We utilized our proposed SAT encoding to address the Anti-Bandwidth Problem, achieving competitive results not only compared to other SAT encoding techniques, but also outperforming robust MIP and CP solvers such as *CPLEX*, *Gurobi*, and *CP-SAT*. By extending our approach from AMO to AMK constraints, we developed a compact representation that minimizes both the number of variables and the generated clauses for AMK cardinality constraints with a ladder structure. Integrating this new encoding technique with other effective SAT encoding and propagation methods shows significant promise for enhancing the performance of practical scheduling problems that involve sequence constraints with ladder structures, such as the Employee Scheduling problem [7], the Nurse Rostering problem [5], and various other scheduling challenges featuring sequence constraints with ladder structures.

Acknowledgements. We thank the authors of Duplex encoding [14] for publishing the source code of Duplex, which allows us to implement the Anti-bandwidth problem more conveniently.

Author Contributions. Hieu Truong Xuan: conceptualization, methodology, formal analysis, investigation, data curation, software, visualization, writing - original draft, review & editing;
Tuyen Kieu Van: methodology, writing - review & editing;
Khanh To Van: conceptualization, investigation, methodology, formal analysis, supervision, funding acquisition, writing - original draft, review & editing.

Funding. This work has been supported by VNU University of Engineering and Technology under project number CN24.10.

Availability of data and materials. The experimental results are available at https://github.com/TruongXuanHieu-H/SCL-Encoding.

Code availability. The source code for these experiments is also available at https://github.com/TruongXuanHieu-H/SCL-Encoding.

Declaration

Ethics approval. Not applicable.

Consent to participate. Not applicable.

Consent for publication. Not applicable.

Conflict of interest. Not applicable.

References

[1] Artigues, C., Hebrard, E., Mayer-Eichberger, V., Siala, M., Walsh, T.: Sat and hybrid models of the car sequencing problem. In: Integration of AI and OR Techniques in Constraint Programming: 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings 11, pp. 268–283 (2014). Springer

[2] Siala, M.: Search, propagation, and learning in sequencing and scheduling problems. PhD thesis, INSA de Toulouse (2015)

[3] Ceschia, S., Dang, N.T.T., De Causmaecker, P., Haspeslagh, S., Schaerf, A.: Second international nurse rostering competition (inrc-ii)—problem description and rules—. arXiv preprint arXiv:1501.04177 (2015)

[4] Kletzander, L., Musliu, N.: Solving the general employee scheduling problem. Computers & Operations Research **113**, 104794 (2020)

[5] Bergman, D., Cire, A.A., Hoeve, W.: Mdd propagation for sequence constraints. Journal of Artificial Intelligence Research **50**, 697–722 (2014)

[6] Siala, M., Hebrard, E., Huguet, M.-J.: An optimal arc consistency algorithm for a particular case of sequence constraint. Constraints **19**, 30–56 (2014)

[7] Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Rollon, E.: Employee scheduling with sat-based pseudo-boolean constraint solving. IEEE access **9**, 142095–142104 (2021)

[8] Lester, M.M.: Pseudo-boolean optimisation for robinx sports timetabling. Journal of Scheduling **25**(3), 287–299 (2022)

[9] Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Quimper, C.-G., Walsh, T.: Reformulating global constraints: The slide and regular constraints. In: Abstraction, Reformulation, and Approximation: 7th International Symposium, SARA 2007, Whistler, Canada, July 18-21, 2007. Proceedings 7, pp. 80–92 (2007). Springer

[10] Sinnl, M.: A note on computational approaches for the antibandwidth problem. Central European Journal of Operations Research **29**(3), 1057–1077 (2021)

[11] Vardi, M.: The sat revolution: Solving, sampling, and counting. In: Mathematical Colloquium (2015)

[12] Haberlandt, A., Green, H., Heule, M.J.: Effective auxiliary variables via structured reencoding. arXiv preprint arXiv:2307.01904 (2023)

[13] Vasconcellos-Gaete, C., Barichard, V., Lardeux, F.: Abacus: A new hybrid encoding for sat problems. In: 2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI), pp. 145–152 (2020). IEEE

[14] Fazekas, K., Sinnl, M., Biere, A., Parragh, S.: Duplex encoding of staircase at-most-one constraints for the antibandwidth problem. In: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pp. 186–204 (2020). Springer

[15] Sinz, C.: Towards an optimal cnf encoding of boolean cardinality constraints. In: International Conference on Principles and Practice of Constraint Programming, pp. 827–831 (2005). Springer

[16] Hieu, T.X., Tuyen, K.V., Khanh, T.V.: Sequential counter encoding for staircase at-most-one constraints. In: Proceedings of the 17th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART, pp. 164–175 (2025). https://doi.org/10.5220/0013124600003890 . SciTePress

[17] Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Mayer-Eichberger, V.: A new look at bdds for pseudo-boolean constraints. Journal of Artificial Intelligence Research **45**, 443–480 (2012)

[18] Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality networks and their applications. In: Theory and Applications of Satisfiability Testing-SAT 2009: 12th International Conference, SAT 2009, Swansea, UK, June 30-July 3, 2009. Proceedings 12, pp. 167–180 (2009). Springer

[19] Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality networks: a theoretical and empirical study. Constraints **16**, 195–221 (2011)

[20] Batcher, K.E.: Sorting networks and their applications. In: Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference, pp. 307–314 (1968)

[21] Bryant, R.E.: Graph-based algorithms for boolean function manipulation. Computers, IEEE Transactions on **100**(8), 677–691 (1986)

[22] Akers: Binary decision diagrams. IEEE Transactions on computers **100**(6), 509–516 (1978)

[23] Chen, J.: A new sat encoding of the at-most-one constraint. Proc. constraint modelling and reformulation, 8 (2010)

[24] Leung, J.Y., Vornberger, O., Witthoff, J.D.: On some variants of the bandwidth minimization problem. SIAM Journal on Computing **13**(3), 650–667 (1984)

[25] Hale, W.K.: Frequency assignment: Theory and applications. Proceedings of the IEEE **68**(12), 1497–1514 (1980)

[26] Cappanera, P.: A survey on obnoxious facility location problems. University of Pisa (1999)

[27] Hu, Y., Gansner, E.R., Kobourov, S.: Visualizing graphs and clusters as maps. IEEE Computer Graphics and Applications **30**(6), 54–66 (2010)

[28] Gent, I.P., Petrie, K.E., Puget, J.-F.: Symmetry in constraint programming. Foundations of Artificial Intelligence **2**, 329–376 (2006)

[29] Biere, A.: Cadical at the sat race 2019. In: Heule, M., Järvisalo, M., Suda, M. (eds.) Proceedings of SAT Race 2019: Solver and Benchmark Descriptions, vol. B-2019-1 of Department of Computer Science Series of Publications B, University of Helsinki 2019, pp. 8–9 (2019)

[30] Rodriguez-Tello, E., Romero-Monsivais, H., Ramírez-Torres, J., Lardeux, F.: Harwell-Boeing Graphs for the CB Problem. https://doi.org/10.13140/2.1.2408.0166

[31] Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard cp modelling language. In: International Conference on Principles and Practice of Constraint Programming, pp. 529–543 (2007). Springer