SIB: Sorted-integers-based Index for Compact and Fast Caching in Top-down Logic Rule Mining

Ruoyu Wang[1], Raymond Wong[1], Daniel Sun[2], and Rajiv Ranjan[3]

[1]University of New South Wales School of Computer Science and Engineering
[2]UGAiForge LLC
[3]Newcastle University

September 03, 2024

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE

SIB: Sorted-integers-based Index for Compact and Fast Caching in Top-down Logic Rule Mining

Ruoyu Wang[1] | **Raymond Wong**[1] | **Daniel Sun**[2] | **Rajiv Ranjan**[3]

[1]Computer Science and Engineering, University of New South Wales, NSW, Australia

[2]UGAiForge LLC., ACT, Australia

[3]Newcastle University, England, United Kingdom

Correspondence
Raymond Wong, University of New South Wales, Sydney, 2032.
Email: ray.wong@unsw.edu.au

Funding Information
This research was supported by the the UNSW School of Computer Science & Engineering scholarship

Abstract

Mining logic rules from structured knowledge bases is the basis of knowledge engineering. Due to the NP-hardness of the rule mining problem, logic rules cannot be efficiently induced from knowledge bases, especially large-scale ones, and most mining techniques employ algorithmic and architectural optimizations to improve efficiency. Data-oriented optimizations have also been explored to some extent, but the data efficiency is relatively low, and the memory consumption is thus becoming a new challenge for state-of-the-art systems. In this article, we propose a compact and efficient index structure for the maintenance of the intermediate data during top-down rule mining. The index is based on a mapping from constant symbols to integers and the sorting of the mapped integers. We evaluate our method on six datasets which contain up to 160K records and are frequently used as benchmarks in knowledge engineering related tasks. The experimental results show that the proposed technique speeds up the rule mining procedure by 5x on average and reduces memory consumption by up to 70%. The space overhead of the data structure is about twice that of the indexed records, which is more than 80% lower than that of the state-of-the-art technique.

KEYWORDS
rule mining, optimization, indexing, data structures, data efficiency, knowledge bases

1 | INTRODUCTION

Logic rule mining provides an interpretable and manipulable way to induce semantic patterns from structured knowledge bases (KBs)[1]. Induced rules can be used to predict missing knowledge[2,3], answer questions/queries[4,5], integrate different knowledge[6,7], and detect information redundancies[8,9,10,11]. However, rule mining algorithms are usually slow and consume a lot of computational resources.

The main reason for the low efficiency of rule mining algorithms is that the complexity of the rule mining problem is NP-hard[10], and the search space of logic rules is exponential to the size of KBs and the maximum length of rules. Although pruning techniques have been extensively proposed[2,12], the number of candidate rules explored during mining routines remains orders of magnitude more than the ones that are qualified and returned[10]. Moreover, the cost for a single evaluation of a specific logic rule is also expensive, especially on large-scale KBs, as the number of variable substitutions in a rule is also exponential to the maximum rule length.

Top-down rule mining techniques have been more extensively optimized and are thus usually faster than bottom-up ones. Although many recent works explore neuro-symbolic approaches to logic rule mining[13,14], these approaches require additional hardware support, such as GPUs, to complete the task. In order to improve the mining efficiency, most top-down techniques optimize the mining algorithm[2,10,15,16] and the implementation architecture[17,18] of the algorithms. Databases and customized data structures have also been exploited to maintain records/tuples in a KB and build indices for fast queries. For example, QuickFOIL[19] implements the FOIL algorithm[20] based on relational database systems and groups queries to avoid repetitive data operations. AMIE[2] employs an in-memory RDF database and builds permutation indices for RDF tuples. SINC[21] uses a

Abbreviations: KB, knowledge base; RDF, resource description framework; ILP, inductive logic programming; SIB, sorted-integer-based; SR, sorted refereće; VC, values in columns; IO, index offsets; NV, numbers of values; LLM, large language model.

TABLE 1 Memory cost (Byte) of rule mining

KB	Fs	Fm	UMLS	CoDEx	WN18	NELL-500
Disk Space	36.44K	148.78K	58.21K	300.43K	1.75M	1.71M
Memory Cost of SINC	32.37M	235.21M	1.43G	6.63G	2.64G	6.90G
Memory Expansion	910	1619	25832	22848	1550	4136
Proportion of Cache (%)	56.49	57.81	73.79	81.13	23.95	64.16

*The memory expansion is the memory cost of SINC over the disk space used by the original KB.

customized data structure to cache intermediate results of logic rule inference, and the inference of other rules may be simply obtained by modifications to the cached content.

Nevertheless, the data efficiency and operation efficiency in such approaches still need improvement. Database systems are efficient for indexing records in KBs, but repeated data queries cannot be fully avoided[10]. The query engines in database systems are optimized for generic data queries rather than logic inference in which *join* queries dominate the query patterns. For example, although customized data structures and the fine-grained specialization operations[12] have been employed in the state-of-the-art system SINC to improve the reuse of intermediate data and simplify data queries to a small set of basic operators, resulting in orders of magnitude acceleration towards previous approaches, the indices of the cache consume too much memory space, especially when mining large KBs. The implementation of SINC with the customized cache structure runs out of memory when mining KBs with millions of records (see Section 5).

Table 1 compares the disk space of the KBs used in Section 5 and the peak memory consumption during rule mining on the KBs by SINC. Rule mining consumes up to four orders of magnitude more space than the original KB. The 5th row in Table 1 shows the proportion of memory consumption by cached content, illustrating that most of memory has been consumed by cached content during rule mining. Therefore, we set the following objectives to study the data efficiency issue in logic rule mining:

- Analyze the data structures used for caching in logic rule mining and propose a compact representation for the cache, such that the memory consumption can be reduced.
- Propose efficient operations on the new data structure, such that the overall rule mining speed on the new data structure will not be slower than the original one.

This article tackles the challenge with a novel index structure based on sorted integer tables. The index blocks also serve as the cache of logic inference in top-down mining procedures, and more efficient data operations can be used to improve the efficiency of overall rule mining. Contributions of this article include:

- We propose a sorted-integer-based (SIB) structure to compactly index the cached records in KBs. References to records in a KB are also used as the cache of logic inference. The structure can be reused and significantly reduces space consumption during rule mining.
- We disassemble the operations used in rule mining into four basic operators: *Get*, *Split*, *Match*, and *Join*, of the proposed index structure. Then, we present efficient algorithms for the operators.
- We comprehensively evaluate our techniques against state-of-the-art rule mining systems and present a detailed ablation study of the data operators based on the new data structure. The results show that the new index structure improves the speed of rule mining by 5x on average and reduces up to 70% of space throughout the rule mining procedures. Besides, the space overhead of the new structure is twice the size of the indexed records, which is much smaller than that of the state-of-the-art technique.

The remainder of the article is as follows: Section 2 reviews techniques for logic rule mining from KBs. Section 3 introduces background knowledge of top-down logic rule mining. The SIB index and the improved cache update operations are proposed in Section 4. Then, we evaluate our technique in Section 5 and present a detailed ablation study. Section 6 discusses the implications and important open questions from this article. Finally, the article is concluded in Section 7.

2 | RELATED WORK

2.1 | Logic Rule Mining

Logic rule mining is mainly studied within the settings of Inductive Logic Programming (ILP)[22], and logic rules can be induced in a discrete or a differentiable manner. The ***discrete approaches*** construct logic rules by explicitly combining predicate and constant symbols appearing in a given KB via a top-down[2,10,19,20] or a bottom-up[15,16] procedure, where rules are modified from general to specific or the inverse, respectively. The top-down methods are usually integrated into database management systems. The bottom-up methods are studied via formal methods and can be employed to generate expressive logic programs, such as answer set programs[23] or higher-order logic programs. The ***differentiable approaches*** represent logic operators as differentiable functions[24,25,26] or train neural networks[13,14] to capture the semantics of logic rules.

2.2 | Optimizations for Top-down Logic Rule Mining

Optimizations of logic rule mining can be categorized as algorithm-, architecture-, or data-oriented. ***Algorithm-oriented*** optimizations improve logic rule mining with more efficient algorithms and procedures. A common solution for efficiency is to restrict the expressiveness of semantic models, such as extracting a fragment of first-order logic where the mining complexity is lowered, such as description logic[27,28], association patterns[8], and frequent graph structures[9]. Pruning is another widely applied methodology to reduce the number of data queries during mining. Besides trivial syntax duplications and renaming duplications[29,30], QuickFOIL[19] also prunes replaceable duplications of literals. AMIE[2] prunes candidate rules that entail an insufficient number of tuples and calculates the confidence score in a lazy manner to stop the calculation of bad scores at an early stage. Statistical estimation has also been explored to improve pruning efficiency[31,32,33,34]. Algorithm-oriented optimizations usually speed up rule mining procedures by 5-10x.

Architectural-oriented optimizations improve the system architecture that supports the rule mining algorithm, and they do not change the behavior of algorithms. Parallelization of logic rule mining belongs to this category. Srinivasan et al.[18] studied the parallelization of ILP algorithms at data and task level. They found the performance improvement via data parallelism can only be expected above some minimal dataset size, and the improvement via task parallelism can only be expected above some minimal search-space size. Nishiyama and Ohwada[17] propose a novel communication protocol for sharing evaluation results of candidate rules, achieving a higher-than-linear speed-up of ILP algorithms. ScaleKB[35] parallelizes the ontological path-finding algorithm to mine first-order Horn rules from open-domain RDF KBs. The system also parallelizes inference procedures to efficiently produce new facts via the induced rules.

Data-oriented approaches focus on optimizing data structures and data operations, not interfering with the control flow of the rule mining algorithm. The proposed technique in this article belongs to this category. FOIL-D[31] integrates the top-down logic rule mining algorithm with relational database systems and converts evaluations of logic rules as SQL queries. CrossMine[36] optimizes data queries used for evaluating the quality of logic rules by an ID propagation technique, reducing the cost of join operations and defining a new grammar to represent the propagation operations. QuickFOIL[19] groups identical SQL operators and runs the queries in batches to avoid repetitive data queries. AMIE[2] also groups similar queries, and the system also calculates quality scores of logic rules in a lazy manner, such that the scores of pruned rules will not be calculated. SINC[21] implements a customized data structure to cache intermediate inference results of logic rules. The systems also propose fine-grained specialization operations[12] to simplify the update of the cache structure. The speed of SINC is faster than the other discrete rule mining approaches, but the memory cost is high, especially when mining large-scale KBs. Our rule mining system employs a more compact index structure that also caches intermediate results, and more efficient operations are applied to the new data structure. Hence, the proposed technique is faster and the memory cost is lower.

3 | PRELIMINARIES

This article focuses on optimizing the discrete top-down mining of first-order Horn rules, the procedure of which is shown in Algorithm 1. In each iteration of top-down mining, every candidate rule in set $\mathbb{R}$ is specialized by introducing new predicates or variables. The quality of each rule r is evaluated by $eval(r)$. The procedure stops when one or more local optimums are found.

Algorithm 1 Top-down Logic Rule Mining

```
Input: RDF KB $\mathcal{K}$
Input: Target relation $p$
Input: Beamwidth $b$
Output: A single logic rule that entails tuples in relation $p$
$\mathbb{R} \leftarrow \{p(?,?) \leftarrow\}$
while $true$ do
    $\mathbb{L} \leftarrow \varnothing$
    for $r \in \mathbb{R}$ and each $r'$ that is specialized from $r$ do
        if $r'$ is not pruned and $eval(r') > eval(r)$ then
            $\mathbb{L} \leftarrow \mathbb{L} \cup \{r'\}$
        end if
    end for
    if $\exists r_m \in \mathbb{R}, s.t. \forall r \in \mathbb{R} \cup \mathbb{L}, eval(r_m) \geq eval(r)$ then
        return $r_m$
    else
        $\mathbb{R} \leftarrow$ $b$ rules in $\mathbb{L}$ with highest scores according to $eval(\cdot)$
    end if
end while
```

TABLE 2 An example knowledge base about kinship relations

uncle		brother		parent	
adam	bob	adam	cain	diana	bob
adam	eric	adam	diana	diana	eric
felix	ginny	felix	hue	hue	ginny
jim	kite	jim	iris	hue	jim

$$uncle(?,?) \leftarrow \tag{1}$$

$$\Downarrow$$

$$uncle(X,?) \leftarrow brother(X,?) \tag{2}$$

$$\Downarrow$$

$$uncle(X,Y) \leftarrow brother(X,?), parent(?,Y) \tag{3}$$

$$\Downarrow$$

$$uncle(X,Y) \leftarrow brother(X,Z), parent(Z,Y) \tag{4}$$

FIGURE 1 An example top-down mining of Rule (4). The parts in red denote the modification by specialization operations.

First-order Horn rules are in the following form: $H \leftarrow B_1, \ldots, B_n$, where H is the head predicate and B_i's are the body predicates. For example:

$$uncle(X,Y) \leftarrow brother(X,Z), parent(W,Y) \tag{5}$$

$$uncle(X,Y) \leftarrow brother(X,?), parent(?,Y) \tag{6}$$

Capital letters in the argument list of a first-order predicate refer to variables, and uncapitalized words are constant symbols. Variables that appear only once (called the *unique variables*) in the rule are simplified as '?'. For example, the above two rules are equivalent.

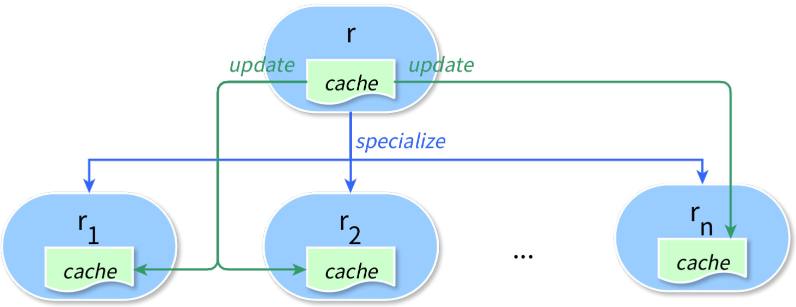

FIGURE 2 Caching in top-down rule mining

Table 2 shows an example kinship KB consisting of three relations: *uncle*, *brother*, and *parent*. Rules (1) to (4) in Figure 1 illustrate a possible top-down mining routine. Rule (4), the induced logic rule, states that if X is the brother of Z and Z is the parent of Y, then we can infer that X is the uncle of Y.

4 | SORTED-INTEGER-BASED INDEX

In order to improve the speed of top-down logic rule mining, the logic inference results of candidate rules should be maintained in memory to reduce the cost of rule quality evaluation. Query grouping[2,19] and result caching[21] are used for this purpose. Caching is better at reusing intermediate data than query grouping, but the memory cost is much higher, especially the cost of indexing the cached records for further operations. In this section, we propose a sorted-integer-based table structure that serves as a compact index as well as a cache for records in KBs.

4.1 | Caching in Logic Rule Mining

As shown in Figure 2, every candidate rule in top-down rule mining manages a cache in memory, explicitly enumerating the substitutions of variables in the rule. When rule r is specialized to $r_1, \dots, r_n$, the cache of r_i is updated from that of r.‘ When a candidate rule is no longer used in the mining procedure, the cache attached to the rule is deleted. The following (referred to as Cache (7)) shows such a cache of Rule (2) with respect to the KB in Table 2:

$$\begin{aligned}
uncle(adam, bob) &\leftarrow brother(adam, cain)\\
uncle(adam, bob) &\leftarrow brother(adam, diana)\\
uncle(adam, eric) &\leftarrow brother(adam, cain)\\
uncle(adam, eric) &\leftarrow brother(adam, diana)\\
uncle(felix, ginny) &\leftarrow brother(felix, hue)\\
uncle(jim, kite) &\leftarrow brother(jim, iris)
\end{aligned} \tag{7}$$

According to the fine-grained specialization[12], every new rule r' is generated by modifying one or two unique variables in the original rule r. When specializing Rule (2) to Rule (3), the second arguments of relations *uncle* and *parent* are assigned to the same variable Y, resulting in a *join* operation by the two arguments. The following shows the cache of Rule (3), where the bold font emphasizes the matched arguments:

$$\begin{aligned}
uncle(adam, \mathbf{bob}) &\leftarrow brother(adam, cain), parent(diana, \mathbf{bob})\\
uncle(adam, \mathbf{bob}) &\leftarrow brother(adam, diana), parent(diana, \mathbf{bob})\\
uncle(adam, \mathbf{eric}) &\leftarrow brother(adam, cain), parent(diana, \mathbf{eric})\\
uncle(adam, \mathbf{eric}) &\leftarrow brother(adam, diana), parent(diana, \mathbf{eric})\\
uncle(felix, \mathbf{ginny}) &\leftarrow brother(felix, hue), parent(hue, \mathbf{ginny})
\end{aligned} \tag{8}$$

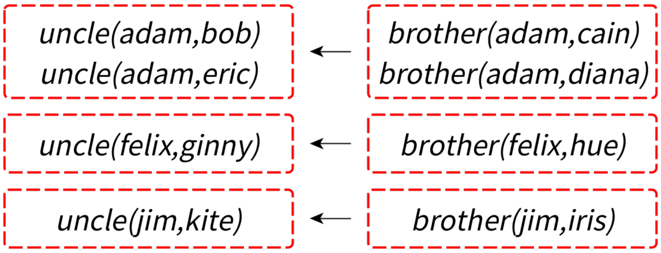

FIGURE 3 Three grouped cache entries representing the six entries in Cache (7), where each dashed red rectangle represents a single block.

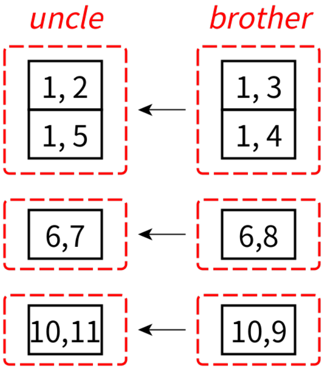

FIGURE 4 Integer tables (shown in dashed red rectangles) representing the cache blocks in Figure 3.

The set of inferred records and the quality score of Rule (3) can thus be efficiently calculated (used in Line 5 of Algorithm 1). Then, the inference results of Rule (4) are generated in the same way from Cache (8). The cache of a specific rule is immutable, as one candidate rule may be specialized to multiple new rules.

4.2 | Sorted-integer-based Index Structure

Theoretically, the number of variable substitutions is exponential to the number of different variables in a logic rule. In order to reduce the number of entries in the cache, the cached records are grouped by relations and variable substitutions. For example, the six entries in Cache (7) can be grouped into three entries, each consisting of two blocks, as shown in Figure 3. The Cartesian product of the two blocks in the first entry of Figure 3 implicitly represents the first four entries in Cache (7). Such blocks in cache entries are the basic components in the rule cache during logic rule mining.

In order to efficiently update the rule cache, SINC[21] builds indices of records in cache blocks according to the argument values, and the indices are implemented by hash tables. However, hash tables are an overkill for this purpose, as they reserve extra memory space for index entry insertion and rearrangement. The extra space in hash tables is unnecessary during logic rule mining, as no modification is applied to existing indices throughout the life cycle of the cache blocks. Taking this property into consideration, we propose a ***sorted-integers-based*** *(SIB)* index structure to compactly index records in cache blocks.

The fist step of building SIB indices is to map the constant symbols in a KB to integers. Without loss of generality, we map all constants to integers $1, 2, \ldots, n$ by a function $I(\cdot)$, where n is the number of constant symbols in the given KB. For the simplicity of presentation, $I(\cdot)$ maps constants to their alphabetical order. I.e., $I(adam) = 1$, $I(bob) = 2, \ldots$. Disregarding the relation symbol, the records in a cache block can be represented as an integer table. For example, the six blocks in Figure 3 can be represented as six integer tables shown in Figure 4.

Given an integer-represented cache block, a SIB index block is created according to the sorted references to the record. Specifically, a SIB index block is composed of four components, as shown in Figure 5: 1) *Sorted References*, 2) *Values in Columns*, 3) *Index Offsets*, and 4) *Numbers of Values*. The construction of a SIB block is illustrated in Algorithm 2.

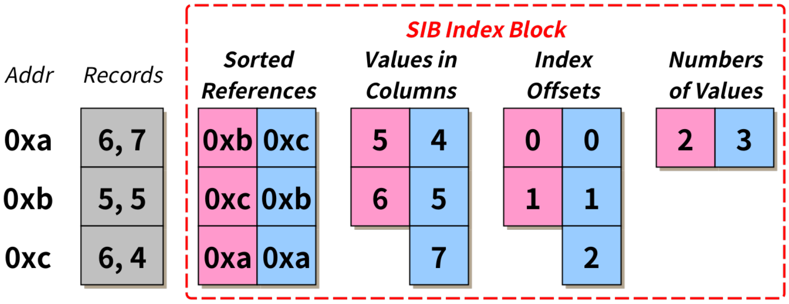

FIGURE 5 The structure of a SIB index block. This figure also illustrates an example of the SIB index block.

Algorithm 2 Construction of an Integer Table

```
Input: A list of records $\mathcal{L}$
Output: A SIB index block $\mathcal{B}$
$\mathcal{B} \leftarrow$ an empty SIB block
for $i = a - 1$ to $0$ ($a$ is the arity of the records in $\mathcal{L}$) do
    $\mathcal{L} \leftarrow$ sort $\mathcal{L}$ w.r.t. the $i^{th}$ argument by a stable sorting algorithm
    $\mathcal{B}.SR[i] \leftarrow$ references of records in $\mathcal{L}$ in the sorted order
    Fill in the other three components corresponding to $\mathcal{B}.SR[i]$ given $\mathcal{L}$
end for
return $\mathcal{B}$
```

- ***Sorted Reference*** *(SR)*: This part contains a columns of references to records in a cache block, where a is the arity of the records. The references in the i^{th} column are sorted in ascending order with respect to the values from the i^{th} to the last arguments of the records. A stable sorting algorithm is used to sort the references, as shown in Algorithm 2. $\mathcal{B}.SR[i][j]$ refers to the element in the i^{th} column and j^{th} row in the SR of a SIB block $\mathcal{B}$.
- ***Values in Columns*** *(VC)*: The i^{th} column in this part contains the integers in the i^{th} arguments of the records without duplication. Values in each column are also in ascending order.
- ***Index Offsets*** *(IO)*: This part indicates where to find the records with specific argument values. Suppose $\mathcal{B}.IO[i][j] = k$, $\mathcal{B}.IO[i][j+1] = k'$, and $\mathcal{B}.VC[i][j] = c$ for a given SIB block $\mathcal{B}$. The references to the records, where the i^{th} arguments are c, start from the k^{th} row to the k'^{th} row (exclusive) in $\mathcal{B}.SR[i]$.
- ***Numbers of Values*** *(NV)*: This part is a list of length a. $\mathcal{B}.NV[i] = l$ means that the lengths of $\mathcal{B}.VC[i]$ and $\mathcal{B}.IO[i]$ are l.

Figure 5 also demonstrates an example SIB index block. On the left of the figure, there are three records from a binary relation. The records are: $(6,7), (5,5), (6,4)$, and the memory addresses of the records are: `0xa, 0xb, 0xc`, respectively. There are two columns in the SR part of the constructed SIB block.

Let $\mathcal{B}$ be the SIB block constructed from the three records. The values in $\mathcal{B}.SR[1]$ (the 2nd column in the "sorted reference" part of $\mathcal{B}$) are the addresses of the three records in ascending order with respect to the 2nd arguments. The values in $\mathcal{B}.SR[0]$ are the addresses of the records in the ascending order w.r.t. the 1st and the 2nd arguments. Values 5 and 6 appear as the first arguments of the records. Thus, $\mathcal{B}.VC[0]$ contains the two values in ascending order. Similarly, $\mathcal{B}.VC[1] = [4,5,7]$. According to the references in $\mathcal{B}.SR[0]$, the value 5 first appears in the first argument of the *first* referred record, which is $(5,5)$. Thus, the corresponding offset $\mathcal{B}.IO[0][0] = 0$. Similarly, $\mathcal{B}.IO[0][1] = 1$, as the value 6 first appears in the first argument of the *second* referred record in $\mathcal{B}.SR[0]$. The fourth part of $\mathcal{B}$, $\mathcal{B}.NV$, is trivial. There are two and three elements in the first and the second columns of $\mathcal{B}.VC$. Therefore, $\mathcal{B}.NV[0] = 2$ and $\mathcal{B}.NV[1] = 3$.

According to the definition, the total space of a SIB block is:

$$S_r \cdot a \cdot |\mathcal{L}| + 2 \cdot O(S_i \cdot a \cdot |\mathcal{L}|) + S_i \cdot a \leq 5S_{\mathcal{L}}$$

Function 3 *Match*

```
Input: $\mathcal{B}$: A SIB block
Input: $i,j$: two arguments in $\mathcal{B}$
Input: $s_i, e_i$: starting and ending offsets (inclusive) of argument $i$ (default: 0, $\mathcal{B}.NV[i]-1$)
Input: $s_j, e_j$: starting and ending offsets (inclusive) of argument $j$ (default: 0, $\mathcal{B}.NV[j]-1$)
Output: The result of operation $Match(\mathcal{B}, i, j)$
$\mathcal{R} \leftarrow \varnothing$
if $s_i > e_i$ or $s_j > e_j$ then
    return $\mathcal{R}$
end if
$o_i \leftarrow \lfloor \frac{e_i - s_i}{2} \rfloor$
$v_i \leftarrow \mathcal{B}.VC[i][o_i]$
$o_j \leftarrow$ the insert index of $v_i$ in $\mathcal{B}.VC[j]$          ▷ Binary search
if $o_j > e_j$ then
    $\mathcal{R} \leftarrow \mathcal{R} \cup Match(\mathcal{B}, i, j, s_i, o_i - 1, s_j, e_j)$
else
    $v_j \leftarrow \mathcal{B}.VC[j][o_j]$
    if $v_i < v_j$ then
        $\mathcal{R} \leftarrow \mathcal{R} \cup Match(\mathcal{B}, i, j, s_i, o_i - 1, s_j, o_j - 1)$
        $\mathcal{R} \leftarrow \mathcal{R} \cup Match(\mathcal{B}, i, j, o_i + 1, e_i, o_j, e_j)$
    else                                                  ▷ $v_i \equiv v_j$
        $\mathcal{R} \leftarrow \mathcal{R} \cup \{r \in Get(\mathcal{B}, i, v_i) | r.i \equiv r.j\}$
        $\mathcal{R} \leftarrow \mathcal{R} \cup Match(\mathcal{B}, i, j, s_i, o_i - 1, s_j, o_j - 1)$
        $\mathcal{R} \leftarrow \mathcal{R} \cup Match(\mathcal{B}, i, j, o_i + 1, e_i, o_j + 1, e_j)$
    end if
end if
return $\mathcal{R}$
```

where S_r, S_i, and $S_{\mathcal{L}}$ are the sizes of reference, integer, and records in $\mathcal{L}$, respectively. The size of a SIB block is no more than five times the total size of the records it indices, as $S_r \leq 2S_i$ in modern operating systems. The equivalence of the above formula can be approached only when $|\mathcal{L}| = 1$. In real-world cases, the space overhead of SIB blocks is only about twice the size of the records, as shown in Section 5.3.

Given that all the records in a cache block are referred to in the corresponding SIB block, it is not necessary to keep the cache block in memory, that is, the SIB blocks also act as cache blocks.

4.3 | Index Update Operations for Logic Rule Mining

Let $\mathcal{L}$ be the list of records maintained by a SIB block $\mathcal{B}$. Operations applied to the cache involved in rule mining are dissembled into four basic operations to SIB blocks:

- *Get*$(\mathcal{B}, i, c)$: Select all records from $\mathcal{L}$, such that the i^{th} arguments of the selected records are of value c;
- *Split*$(\mathcal{B}, i)$: Split $\mathcal{L}$ into s slices, such that s is minimum, and in each slice, the i^{th} arguments of the records are the same;
- *Match*$(\mathcal{B}, i, j)$: Find all records that the i^{th} and the j^{th} arguments are the same, then *Split* such records by the i^{th} arguments.
- *Join*$(\mathcal{B}_1, i_1, \mathcal{B}_2, i_2)$: θ-join two blocks $\mathcal{B}_1$ and $\mathcal{B}_2$ by the two arguments and *Match* the result by the arguments:

$$Match(SIB(\mathcal{L}_1 \bowtie_{\mathcal{L}_1.i_1=\mathcal{L}_2.i_2} \mathcal{L}_2), \mathcal{L}_1.i_1, \mathcal{L}_2.i_2)$$

where $SIB(\mathcal{L})$ means constructing a SIB block from $\mathcal{L}$.

Function 4 *Join*

```
Input: $\mathcal{B}_1, i_1$: The first SIB block and the argument in the block;
Input: $\mathcal{B}_2, i_2$: The second SIB block and the argument in the block;
Input: $s_1, e_1$: The starting and ending offsets (inclusive) of argument $\mathcal{B}_1.i_1$ (default: 0,
    $\mathcal{B}_1.NV[i_1] - 1$)
Input: $s_2, e_2$: The starting and ending offsets (inclusive) of argument $\mathcal{B}_2.i_2$ (default: 0,
    $\mathcal{B}_2.NV[i_2] - 1$)
Output: The result of operation $Join(\mathcal{B}_1, i_1, \mathcal{B}_2, i_2)$
$\mathcal{R} \leftarrow \varnothing$
if $s_1 > e_1$ or $s_2 > e_2$ then
    return $\mathcal{R}$
end if
$o_1 \leftarrow \lfloor \frac{e_1 - s_1}{2} \rfloor$
$v_1 \leftarrow \mathcal{B}_1.VC[i_1][o_1]$
$o_2 \leftarrow$ the insert index of $v_1$ in $\mathcal{B}_2.VC[i_2]$                    ▷ Binary search
if $o_2 > e_2$ then
    $\mathcal{R} \leftarrow \mathcal{R} \cup Join(\mathcal{B}_1, i_1, s_1, o_1 - 1, \mathcal{B}_2, i_2, s_2, e_2)$
else
    $v_2 \leftarrow \mathcal{B}_2.VC[i_2][o_2]$
    if $v_1 < v_2$ then
        $\mathcal{R} \leftarrow \mathcal{R} \cup Join(\mathcal{B}_1, i_1, \mathcal{B}_2, i_2, s_1, o_1 - 1, s_2, o_2 - 1)$
        $\mathcal{R} \leftarrow \mathcal{R} \cup Join(\mathcal{B}_1, i_1, \mathcal{B}_2, i_2, o_1 + 1, e_1, o_2, e_2)$
    else                                                                        ▷ $v_1 \equiv v_2$
        $\mathcal{R} \leftarrow \mathcal{R} \cup \{Get(\mathcal{B}_1, i_1, v_1) \bowtie Get(\mathcal{B}_2, i_2, v_2)\}$
        $\mathcal{R} \leftarrow \mathcal{R} \cup Join(\mathcal{B}_1, i_1, \mathcal{B}_1, i_2, s_1, o_1 - 1, s_2, o_2 - 1)$
        $\mathcal{R} \leftarrow \mathcal{R} \cup Join(\mathcal{B}_1, i_1, \mathcal{B}_1, i_2, o_1 + 1, e_1, o_2 + 1, e_2)$
    end if
end if
return $\mathcal{R}$
```

All the above operations can be efficiently implemented under the schema of the SIB index. *Get* queries can be answered by a binary search w.r.t. the i^{th} column in VC. *Split* queries scan through $\mathcal{B}.VC[i]$. Results of *Match* and *Join* queries are obtained by the divide-and-search procedures shown in Algorithms 3 and 4. When matching $\mathcal{B}.VC[i]$ with $\mathcal{B}.VC[j]$, the median value in $\mathcal{B}.VC[i]$ is used to divide $\mathcal{B}.VC[i]$ and $\mathcal{B}.VC[j]$ into four smaller arrays. Then, the matching can be recursively calculated by matching those small arrays, as the arrays are already sorted. *Join* is implemented in a similar way.

The worst-case complexity of the search procedure is $O(nlog(n))$, and in the best case, $\Omega(log(n))$, where n is the number of records in $\mathcal{L}$. Figure 6 shows an example of $Join(\mathcal{B}_1, i_1, \mathcal{B}_2, i_2)$, where $\mathcal{B}_1.VC[i_1]$ = $[1, 5, 8, 10, 15]$ and $\mathcal{B}_2.VC[i_2]$ = $[2, 3, 4, 6, 8]$. The *Join* based on the SIB block uses 3 binary searches to find the final result, while by hash tables, 5 table searches are required. The difference in the number of searches will be more significant on larger KBs.

The first column in the sorted reference part of a SIB block $\mathcal{B}$, i.e., $\mathcal{B}.SR[0]$, can be used to check the existence of a record in $\mathcal{L}$ via a binary search, as all records in $\mathcal{B}.SR[0]$ are in the alphabetical order, which is guaranteed by Proposition 1.

Proposition 1. *All records in a SIB block are of alphabetical order with respect to the references in the first column in SR of the generated SIB block.*

Proof. According to Algorithm 2, the records in $\mathcal{L}$ are sorted in alphabetical order with respect to the last column after the first iteration of sorting.

Moreover, if the records in $\mathcal{L}$ are in alphabetical order with respect to the i^{th} to the last arguments before the i^{th} iteration, the records in $\mathcal{L}$ are in alphabetical order with respect to the $(i-1)^{\text{th}}$ to the last arguments after the iteration, as the sorting algorithm is stable.

Therefore, all records in $\mathcal{L}$ are of alphabetical order with respect to the references in the 1^{st} column in SR of the generated SIB block. □

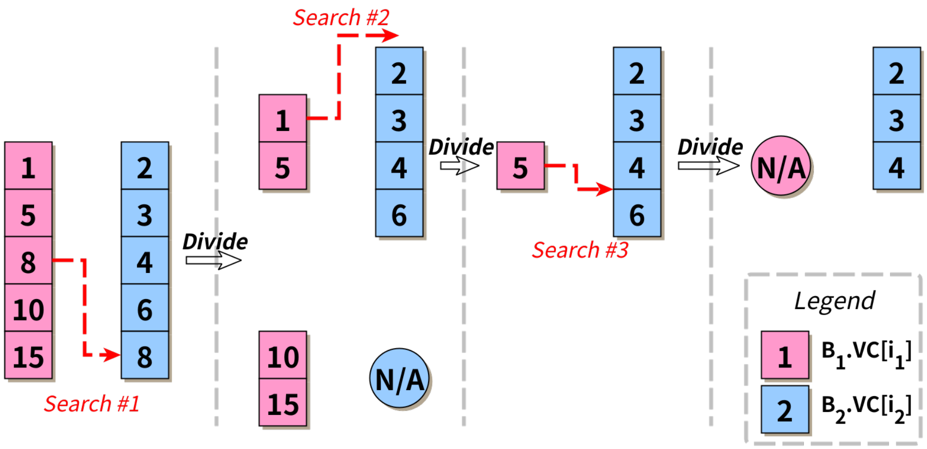

FIGURE 6 An example of $Join(\mathcal{B}_1, i_1, \mathcal{B}_2, i_2)$

The following shows the query results of the example SIB block $\mathcal{B}$ in Figure 5:

- $Get(\mathcal{B}, 0, 6) = [(6, 4), (6, 7)]$
- $Split(\mathcal{B}, 0) = \{[(5, 5)], [(6, 4), (6, 7)]\}$
- $Match(\mathcal{B}, 0, 1) = \{[(5, 5)]\}$
- $Join(\mathcal{B}, 0, \mathcal{B}, 1) = \{[((5, 5), (5, 5))]\}$

5 | EVALUATION

This section evaluates the proposed indexing technique and answers the following questions:

Q1 How fast and compact is the proposed sorted-integer-based index structure compared to state-of-the-art systems with respect to overall logic rule mining?
Q2 To what extent are the construction and query operations optimized by the SIB index structure?
Q3 Is the proposed sorted-integer-based index structure robust to the mapping of constant symbols in speed and space?

5.1 | Experimental Settings

Datasets: Table 3 shows the statistics of the datasets used in our experiments. The datasets are from different domains of knowledge and are frequently used as benchmarks in related studies. “Fs”, “Fm”, “UMLS”, and “WN18” are adopted in SINC[12]. CoDEx is a benchmark for knowledge graph completion tasks[37]. “NELL-500” is a benchmark extracted from “NELL” (used in[9]) by filtering out relations containing less than 500 tuples. “Yago”[38] and “DBpedia”[39] are large-scale knowledge graphs that are widely used as benchmarks of knowledge-related tasks. The small and medium-sized datasets are included in our GitHub project[‡].

Competitors: Our rule mining system with the proposed indexing technique is **SIB**. The baseline is **SINC**[12], as it is the state-of-the-art top-down mining system for first-order logic rules. The rule mining speed and the memory usage of SINC is compared to SIB. The latest version of **AMIE**[2] is also adopted as the baseline of rule mining speed. Most differentiable logic rule mining techniques rely on more hardware (e.g., GPUs) and parallelization to train the models, and the resource consumption is higher than the discrete ones. In this article, our goal is to optimize data structures for top-down logic rule mining under a single thread. Therefore, the differentiable techniques are not adopted as competitors.

[‡] https://github.com/TramsWang/SIB/tree/main

TABLE 3 Dataset overview

Dataset	Short	#Rel.	#Const.	#Rec.
Family.simple	Fs	4	402	1200
Family.medium	Fm	9	702	5500
UMLS	U	46	135	6529
CoDEx	C	42	2034	37K
WN18	W	18	41K	151K
NELL-500	N	122	33K	167K
Yago	Y	37	2M	4M
DBpedia	D	620	4M	20M

*The 3rd, 4th, and 5th columns are the numbers of relations, constant symbols, and records in the KBs.

Large language models (LLMs), such as ChatGPT[§], Bard[¶] [40], FLAN[41], and LLaMA[42], have been used for few-shot learning tasks, such as knowledge-based question answering[43,44]. Although LLMs have been criticized for their induction and reasoning capabilities[45], researchers have been trying to convert the rule mining task into a few-shot learning manner[46]. Despite this, LLMs are not appropriate to be competitors for our techniques. First, the goal of this article is to reduce the computational cost of overall rule mining. The model size itself of an LLM is much larger than the memory cost throughout the rule mining of the competitors in this experimental evaluation. Second, the inference of LLMs relies on hardware clusters, especially GPUs, which is much heavier than the discrete mining involved in this article. Therefore, LLMs are not adopted as competitors.

Environment: The proposed approach is implemented in C++ and is open-sourced on GitHub. As we are not focusing on multi-threading in this article, the systems are implemented and compared in a single-thread manner. The beamwidth of top-down rule mining is 3, and the compression capacity[11] is used as the $eval(\cdot)$ function in Algorithm 1. The same setting is applied to SINC. The maximum rule length of AMIE is set to 5. All tests were performed on Deepin Linux (kernel: 5.18.4-amd64) with Ryzen 3600X and 128GB RAM.

Measures: Rule mining time and memory consumption are measured to compare the performance of the proposed index structure. The resulting rule quality (e.g., accuracy) is not compared, as the proposed data structure does not interfere with the workflow of the rule mining algorithm and thus has no effect on the rule quality.

5.2 | SIB Index vs State-of-the-art

This section answers the research question Q1: The rule mining has been accelerated by about 5x on average compared to SINC and more than 10x compared to AMIE; In terms of memory, up to 70% memory space can be reduced.

Table 4 compares the rule mining time of SIB against SINC and AMIE. Runtime errors occurred when mining "UMLS" and "NELL-500" by AMIE, which is denoted as × in Table 4. SINC runs out of memory on "Yago" and "DBpedia". SINC is much faster than AMIE, and SIB is about 5x faster than SINC on average. One reason for such acceleration is that less memory is acquired and accessed by SIB. As presented in Table 5, where the peak memory consumption throughout a rule mining procedure is shown, up to 72% of memory has been reduced. Another reason is the speed-up by the query operations on the SIB index, and this is more significant on datasets where the average numbers of records and constants in a relation are large. For example, the speed-up of SIB is up to 8.86x faster than SINC on "WN18", while the number is 1.82 on "UMLS". The detailed analyses for these reasons are studied in the next section.

5.3 | Ablation Study

This section answers the research question Q2: The speed-up of SIB is mainly due to the efficient construction and *Join* queries on the proposed index structure; The construction of SIB indices reduces up to 85% memory space and is 2x faster than the hash tables in SINC; *Join* queries to the SIB index blocks is up to 4.5x faster than SINC, and the speed-up is more significant on relations containing more constants.

[§] https://openai.com/blog/chatgpt

[¶] https://bard.google.com

TABLE 4 Rule mining speed

Dataset	Overall Time (s)			Speed-up towards	
	SIB	SInc	AMIE	SInc	AMIE
Fs	0.06	0.39	2.30	6.17	36.51
Fm	1.49	10.37	20.35	6.94	13.69
UMLS	20.30	36.89	×	1.82	-
CoDEx	66.32	395.37	$1h$	5.96	54.28
WN18	49.58	439.17	$> 3h$	8.86	> 217.83
NELL-500	544.63	4424.65	×	8.12	-
Yago	1.2h	×	×	-	-
DBpedia	7.9h	×	×	-	-

TABLE 5 Peak memory cost (GByte) throughout rule mining

Dataset	SIB	SInc	Memory Reduction (%)
Fs	0.01	0.03	71.62
Fm	0.06	0.23	72.19
UMLS	1.10	1.43	23.30
CoDEx	5.85	6.63	11.81
WN18	1.02	2.64	61.42
NELL-500	2.30	6.90	66.69
Yago	19.86	×	-
DBpedia	7.88	×	-

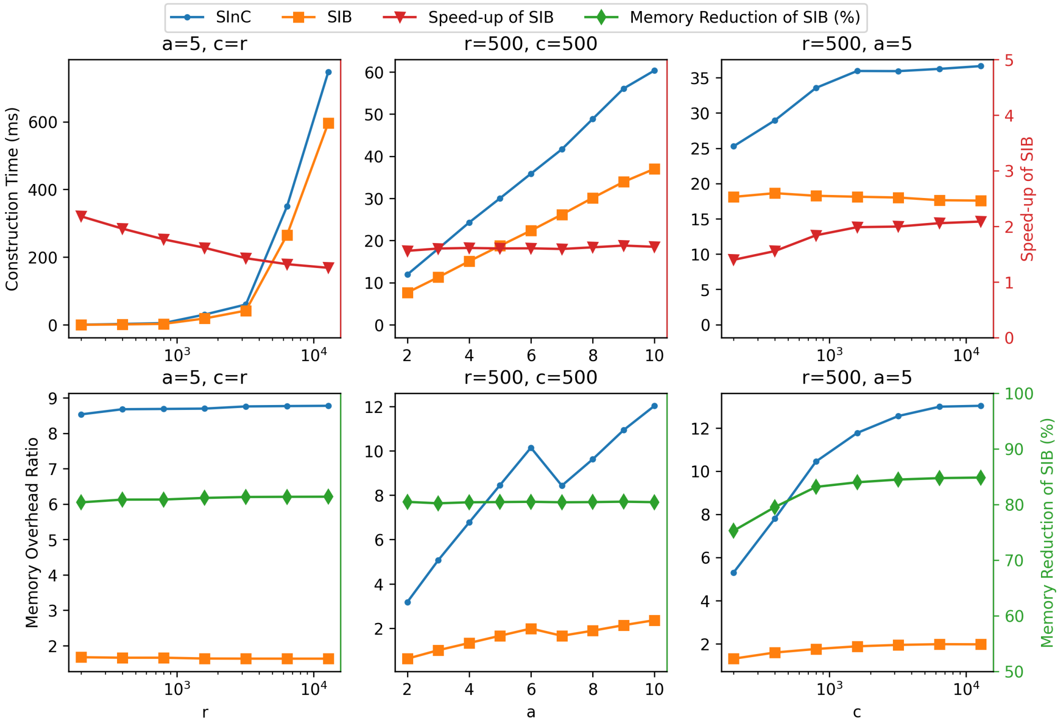

FIGURE 7 Memory overhead of the SIB index vs SInc

In order to explain the detailed reasons for the performance of the SIB index, we generate random datasets with different numbers of records (r), record arities (a), and the number of constants in the records (c). Experiments have been repeated 100 times for each group of the settings. The total time consumption and memory overhead of index construction is presented in Figure 7. The three columns correspond to the settings varying the three parameters, and the two rows refer to the time cost and

TABLE 6 Proportion (%) of queries during rule mining in SInc

Dataset	**Fs**	**Fm**	**UMLS**	**CoDEx**	**WN18**	**NELL-500**
Get	34.04	26.19	27.17	62.79	19.93	17.04
Split	0	2.06×10^{-2}	9.72×10^{-3}	2.19×10^{-2}	1.08×10^{-2}	3.27×10^{-3}
Match	4.65×10^{-3}	1.08×10^{-3}	4.37×10^{-4}	8.62×10^{-5}	1.56×10^{-4}	1.59×10^{-4}
Join	65.96	73.79	72.82	37.18	80.06	82.96
Join (Time)	99.97	99.96	99.97	99.88	99.98	99.99

*The second to the fourth rows show the proportions of the queries with respect to the total number of queries. The last row shows the proportion of *Join* query time with respect to the total time cost of all queries on the index.

memory overhead of index construction in SIB and SInc. The memory overhead is given by the memory cost of the index over that of the original records in the tested relation:

$$\text{Memory Overhead Ratio} = \frac{\text{Index Data Memory Cost}}{\text{Records Data Memory Cost}} \tag{9}$$

SIB outperforms SInc in all settings. The construction speed is about 2x faster than the hash tables, as less memory is modified. The reduction of memory consumption is more significant (up to 85%) when there are more constants in the tested relations. Although the performance of SIB and SInc are close in the upper left sub-figure, the construction time of SIB is still less than SInc.

Figure 8 compares the query time in SIB and SInc under the same settings as Figure 7. Each row in the figure shows the total time of one type of queries on the index. The results show that the *Join* query on the SIB index is more than 50% faster than SInc, and for relations with more constants, the acceleration is more significant (more than 4.5x faster, shown in the bottom right sub-figure). This is because the divide-and-search *Join* of SIB significantly reduces the number of searches, as analyzed in Section 4.3. For example, in the experiments when $r = 500$, $a = 5$, and $c = 500$, the number of the constant search in SIB is 46.34% less than that of SInc. The *Get* and *Split* queries in SIB are about 15% slower than SInc. *Match* queries are about 70% slower than SInc, but the difference narrows down when the number of constants in a relation is increasing. Despite the fact that these queries are slower, the costs of the operations do not dominate the rule mining procedure. As shown in Table 6, the proportions of *Join* queries are up to 80% with respect to the number of all queries during rule mining routines, and the proportions are more than 99% with respect to the total query time. Therefore, the overall speed of SIB is faster than SInc.

5.4 | Robustness to Integer Mappings

This section answers the research question Q3: The difference in rule mining speed under different mapping functions of constants is less than 5% on average, and the difference in memory consumption is less than 1%; Therefore, the SIB index is robust to the constant mapping strategy.

In this section, we employ four strategies for the mapping function $I(\cdot)$ and test the overall rule mining speed:

- **Original**: Mapping constant symbols to the order they appear in a KB;
- **Alphabetical**: Mapping constants to their alphabetical order;
- **Frequency**: Mapping constants to their frequency order (decreasing);
- **Random**: Mapping constants to a random order.

Let s denote a strategy among the above, T_s be the overall rule mining time with the mapping function under s. The variation of the mining time is measured by (variation of mining memory is calculated in the same way):

$$\frac{\max_s T_s - \min_s T_s}{\text{avg}_s T_s} \times 100\% \tag{10}$$

The results are shown in Table 7. The rule mining time costs on the same KB are close to each other, and the maximum performance variation is 5.63%. Different order of constants may affect individual *Join* queries. Some queries are slower, while

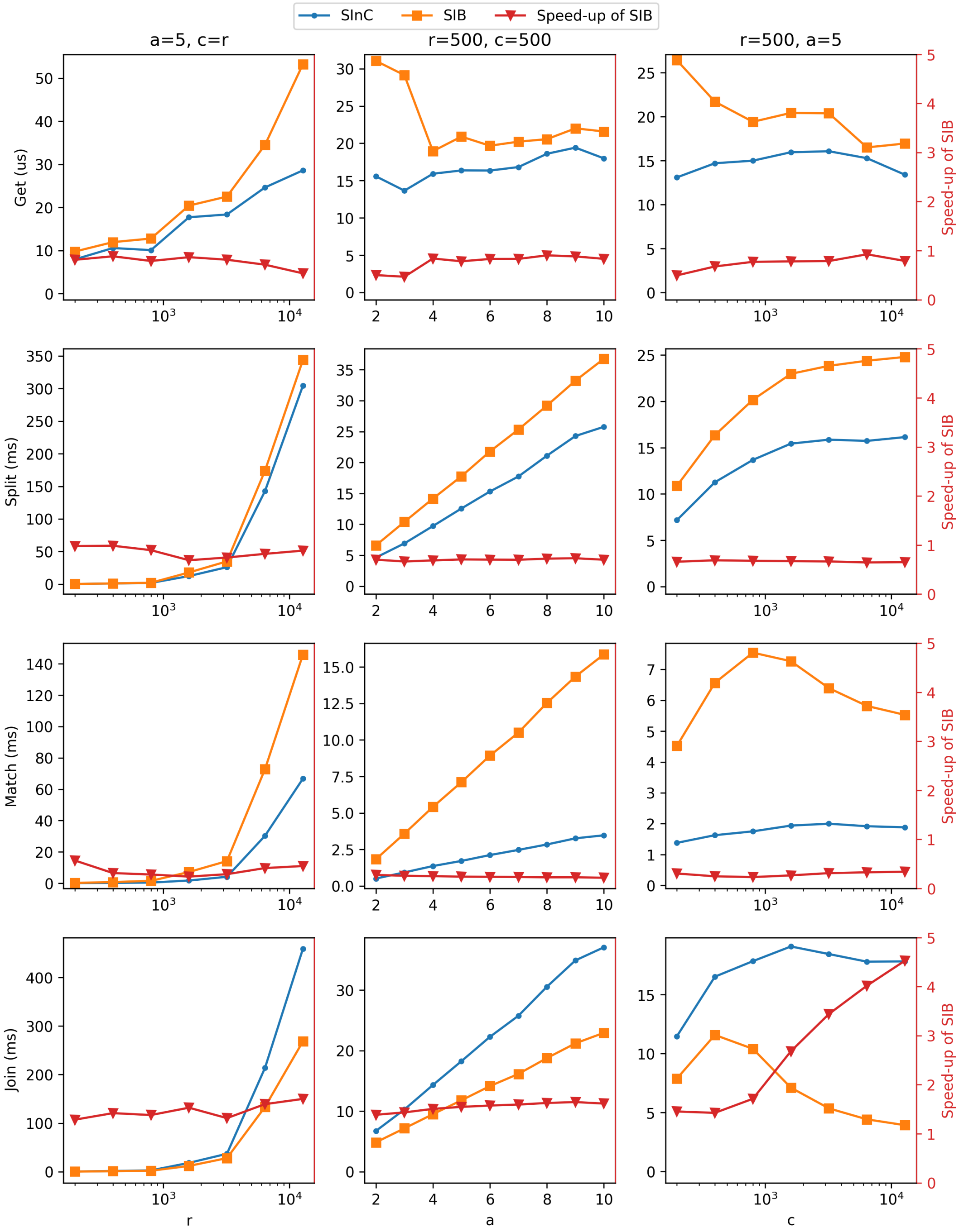

FIGURE 8 Query time of the SIB index vs SInC

others are faster. The overall time is insensitive to such variations, as all records in a KB are accessed during rule mining. According to the definition of the SIB index, the mapping of constants does not affect the index structure. Thus, the size of indices is the same under different mapping strategies, which is justified in Table 8. The difference of space costs is due to temporary variables used in index queries and the memory maintenance by the underlying operating system, and the variation is less than 1% in most datasets.

TABLE 7 Rule mining time (s) with different strategy of function $I(\cdot)$

Dataset	**Constant Mapping by the Order of**				**Variation (%)**
	Original	Alphabetical	Frequency	Random	
Fs	0.06	0.06	0.06	0.06	4.80
Fm	1.49	1.42	1.42	1.50	5.63
UMLS	20.30	20.55	20.15	20.44	1.95
CoDEx	66.32	66.07	64.09	66.30	3.39
WN	49.58	49.63	48.62	50.77	4.33
NELL-500	544.63	551.89	550.57	558.05	2.44
Yago	1.2h	1.2h	1.2h	1.2h	2.63
DBpedia	7.9h	7.8h	7.8h	7.9h	3.61

TABLE 8 Rule mining memory (MByte) with different strategy of function $I(\cdot)$

Dataset	**Constant Mapping by the Order of**				**Variation (%)**
	Original	Alphabetical	Frequency	Random	
Fs	8.97	8.74	8.84	8.94	2.59
Fm	63.87	63.81	63.82	63.73	0.22
UMLS	1126.40	1126.42	1126.31	1126.41	0.01
CoDEx	5995.45	5987.07	5989.92	5988.35	0.14
WN	1044.21	1043.60	1044.36	1044.28	0.07
NELL-500	2354.89	2354.95	2355.73	2354.92	0.04
Yago	19.86G	19.85G	19.85G	19.86G	0.03
DBpedia	7.88G	7.88G	7.89G	7.89G	0.12

6 | DISCUSSION

Most studies on logic rule mining, especially the ones under the discrete approaches, focus on optimizations from the algorithmic and architectural perspectives rather than system implementation and data efficiency, even though the practice and experience in other domains of data mining have shown the importance of data-oriented optimizations[47,48,49,50].

Algorithm-oriented optimizations improve rule mining speed with a more efficient control flow. Architecture-oriented optimizations improve system architectures supporting mining algorithms. In this article, we improve the space efficiency of the underlying data structures of the discrete top-down mining of first-order Horn rules. Compared to the results of prior works in Section 2, the speed-up of data-oriented approaches is competitive to the algorithm- and architecture-oriented ones, implying that data-oriented optimizations are as critical as the other two methodologies to logic rule mining.

As reported in previous works, the hardware requirement when mining KBs with hundreds of thousands of predicates is either a cluster consisting of tens of computers or dozens of threads with more than 100GB memory. AnyBURL[51] extends the bottom-up mining process to millions of tuples in large-scale knowledge graphs, but it requires 1TB memory to complete the task. Our work reduces the amount of computational resources to a single thread with no more than 10GB memory, which is easily met on a normal PC platform.

Although the proposed technique is effective to top-down logic rule mining, whether the data structures are applicable to bottom-up approaches requires further explorations. Recently, differentiable mining approaches of logic rules have frequently been studied, and most of them are by neural networks[14,25,52,53]. These approaches rely on GPUs due to the implementation of low-level frameworks. To what extent data efficiency impacts the performance of such approaches and how it can be improved should also be explored in the future.

7 | CONCLUSION

In this article, we improve the data efficiency of discrete top-down logic rule mining via a compact and efficient index structure. The index is based on mapping constants to integers and sorting the records with respect to the mapped integers. The overall results show a 5x speed-up and a 70% total memory cost reduction. The reasons are explained in a detailed ablation study. We also show that the approach is robust to different strategies of the constant mapping function. Most importantly, the results emphasize the importance of data efficiency in the discrete top-down mining of logic rules. In the future, we will explore the data efficiency of differentiable rule mining methods and extend the study to multi-dimensional relations.

AUTHOR CONTRIBUTIONS

Ruoyu Wang: Conceptualization, Methodology, Software, Validation, Data Curation, Writing - Original Draft, Visualization; **Raymond Wong:** Investigation, Resources, Writing - Review & Editing, Supervision, Project Administration, Funding Acquisition; **Daniel Sun:** Writing - Review & Editing, Supervision, Project Administration.

ACKNOWLEDGMENTS

This research is supported by the UNSW School of Computer Science & Engineering scholarship.

FINANCIAL DISCLOSURE

None reported.

CONFLICT OF INTEREST

The authors declare no potential conflict of interests.

REFERENCES

1. Chen P, Du X, Lu Z, Wu J, Hung PCK. EVFL: An explainable vertical federated learning for data-oriented Artificial Intelligence systems. *J. Syst. Archit.*. 2022;126:102474. doi: 10.1016/J.SYSARC.2022.102474
2. Lajus J, Galárraga L, Suchanek FM. Fast and Exact Rule Mining with AMIE 3. In: . 12123 of *Lecture Notes in Computer Science*. Springer 2020:36–52
3. Park E, Wong RK, Chu VW. Story Ending Generation Using Commonsense Casual Reasoning and Graph Convolutional Networks. In: Gal K, Nowé A, Nalepa GJ, Fairstein R, Radulescu R., eds. *ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland - Including 12th Conference on Prestigious Applications of Intelligent Systems (PAIS 2023)*. 372 of *Frontiers in Artificial Intelligence and Applications*. IOS Press 2023:1843–1850
4. Cui W, Xiao Y, Wang H, Song Y, Hwang S, Wang W. KBQA: Learning Question Answering over QA Corpora and Knowledge Bases. *Proc. VLDB Endow.*. 2017;10(5):565–576. doi: 10.14778/3055540.3055549
5. Li L, Zhang H, Fang Z, Xie Z, Liu J. Transductive Cross-Lingual Scene-Text Visual Question Answering. In: Luo B, Cheng L, Wu Z, Li H, Li C., eds. *Neural Information Processing - 30th International Conference, ICONIP 2023, Changsha, China, November 20-23, 2023, Proceedings, Part VI*. 14452 of *Lecture Notes in Computer Science*. Springer 2023:452–467
6. Huang Q, Yuan Z, Xing Z, Zuo Z, Wang C, Xia X. 1+1>2: Programming Know-What and Know-How Knowledge Fusion, Semantic Enrichment and Coherent Application. *IEEE Trans. Serv. Comput.*. 2023;16(3):1540–1554. doi: 10.1109/TSC.2022.3207273
7. Shu Y, Zhang J, Huang G, Chi C, He J. Entity alignment via graph neural networks: a component-level study. *World Wide Web (WWW)*. 2023;26(6):4069–4092. doi: 10.1007/S11280-023-01221-8
8. Joshi AK, Hitzler P, Dong G. Logical Linked Data Compression. In: Cimiano P, Corcho Ó, Presutti V, Hollink L, Rudolph S., eds. *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*. 7882 of *Lecture Notes in Computer Science*. Springer 2013:170–184
9. Belth C, Zheng X, Vreeken J, Koutra D. What is Normal, What is Strange, and What is Missing in a Knowledge Graph: Unified Characterization via Inductive Summarization. In: ACM / IW3C2 2020:1115–1126
10. Wang R, Sun D, Wong RK. Symbolic Minimization on Relational Data. *IEEE Trans. Knowl. Data Eng.*. 2023;35(9):9307–9318. doi: 10.1109/TKDE.2022.3222827
11. Wang R, Sun D, Wong RK, Ranjan R, Zomaya AY. SInC: Semantic approach and enhancement for relational data compression. *Knowl. Based Syst.*. 2022;258:110001. doi: 10.1016/J.KNOSYS.2022.110001
12. Wang R, Sun D, Wong RK. RDF Knowledge Base Summarization by Inducing First-Order Horn Rules. In: Amini M, Canu S, Fischer A, Guns T, Novak PK, Tsoumakas G., eds. *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2022, Grenoble, France, September 19-23, 2022, Proceedings, Part II*. 13714 of *Lecture Notes in Computer Science*. Springer 2022:188–204
13. Sen P, Carvalho dBWSR, Riegel R, Gray AG. Neuro-Symbolic Inductive Logic Programming with Logical Neural Networks. In: AAAI Press 2022:8212–8219
14. Yang Y, Song L. Learn to Explain Efficiently via Neural Logic Inductive Learning. In: OpenReview.net 2020.
15. Muggleton SH, Santos JCA, Tamaddoni-Nezhad A. ProGolem: A System Based on Relative Minimal Generalisation. In: Raedt LD., ed. *Inductive Logic Programming, 19th International Conference, ILP 2009, Leuven, Belgium, July 02-04, 2009. Revised Papers*. 5989 of *Lecture Notes in Computer Science*. Springer 2009:131–148
16. Muggleton SH, Lin D, Pahlavi N, Tamaddoni-Nezhad A. Meta-interpretive learning: application to grammatical inference. *Mach. Learn.*. 2014;94(1):25–49. doi: 10.1007/S10994-013-5358-3
17. Nishiyama H, Ohwada H. Parallel Inductive Logic Programming System for Superlinear Speedup. In: Lachiche N, Vrain C., eds. *Inductive Logic Programming - 27th International Conference, ILP 2017, Orléans, France, September 4-6, 2017, Revised Selected Papers*. 10759 of *Lecture Notes in Computer Science*. Springer 2017:112–123
18. Srinivasan A, Faruquie TA, Joshi S. Data and task parallelism in ILP using MapReduce. *Mach. Learn.*. 2012;86(1):141–168. doi: 10.1007/S10994-011-5245-8
19. Zeng Q, Patel JM, Page D. QuickFOIL: Scalable Inductive Logic Programming. *Proc. VLDB Endow.*. 2014;8(3):197–208. doi: 10.14778/2735508.2735510
20. Quinlan JR. Learning Logical Definitions from Relations. *Mach. Learn.*. 1990;5:239–266. doi: 10.1007/BF00117105

21. Wang R, Sun D, Wong RK, Ranjan R. Horn rule discovery with batched caching and rule identifier for proficient compressor of knowledge data. *Softw. Pract. Exp.*. 2023;53(3):682–703. doi: 10.1002/SPE.3165
22. Cropper A, Dumancic S, Muggleton SH. Turning 30: New Ideas in Inductive Logic Programming. In: Bessiere C., ed. *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*ijcai.org 2020:4833–4839
23. Law M, Russo A, Broda K. Inductive Learning of Answer Set Programs. In: Fermé E, Leite J., eds. *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*. 8761 of *Lecture Notes in Computer Science*. Springer 2014:311–325
24. Raedt LD, Kersting K. Probabilistic Inductive Logic Programming. In: Ben-David S, Case J, Maruoka A., eds. *Algorithmic Learning Theory, 15th International Conference, ALT 2004, Padova, Italy, October 2-5, 2004, Proceedings*. 3244 of *Lecture Notes in Computer Science*. Springer 2004:19–36
25. Evans R, Grefenstette E. Learning Explanatory Rules from Noisy Data. *J. Artif. Intell. Res.*. 2018;61:1–64. doi: 10.1613/JAIR.5714
26. Raedt LD, Dries A, Thon I, Broeck dGV, Verbeke M. Inducing Probabilistic Relational Rules from Probabilistic Examples. In: Yang Q, Wooldridge MJ., eds. *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*AAAI Press 2015:1835–1843.
27. Kietz J. Learnability of Description Logic Programs. In: . 2583 of *Lecture Notes in Computer Science*. Springer 2002:117–132
28. Fanizzi N, d'Amato C, Esposito F. DL-FOIL Concept Learning in Description Logics. In: . 5194 of *Lecture Notes in Computer Science*. Springer 2008:107–121
29. Costa VS, Srinivasan A, Camacho R, et al. Query Transformations for Improving the Efficiency of ILP Systems. *The Journal of Machine Learning Research.* 2003;4:465–491.
30. Fonseca NA, Costa VS, Silva FMA, Camacho R. On Avoiding Redundancy in Inductive Logic Programming. In: . 3194 of *Lecture Notes in Computer Science*. Springer 2004:132–146
31. Bockhorst J, Ong IM. FOIL-D: Efficiently Scaling FOIL for Multi-relational Data Mining of Large Datasets. In: Camacho R, King RD, Srinivasan A., eds. *Inductive Logic Programming, 14th International Conference, ILP 2004, Porto, Portugal, September 6-8, 2004, Proceedings*. 3194 of *Lecture Notes in Computer Science*. Springer 2004:63–79
32. Kononenko I, Robnik-Sikonja M, Pompe U. ReliefF for estimation and discretization of attributes in classification, regression, and ILP problems. *Artificial intelligence: methodology, systems, applications.* 1996:31–40.
33. Oliphant L, Shavlik JW. Using Bayesian Networks to Direct Stochastic Search in Inductive Logic Programming. In: . 4894 of *Lecture Notes in Computer Science*. Springer 2007:191–199
34. Pitangui CG, Zaverucha G. Learning Theories Using Estimation Distribution Algorithms and (Reduced) Bottom Clauses. In: . 7207 of *Lecture Notes in Computer Science*. Springer 2011:286–301
35. Chen Y, Wang DZ, Goldberg S. ScaLeKB: scalable learning and inference over large knowledge bases. *VLDB Journal.* 2016;25(6):893–918. doi: 10.1007/S00778-016-0444-3
36. Yin X, Han J, Yang J, Yu PS. Efficient Classification across Multiple Database Relations: A CrossMine Approach. *IEEE Trans. Knowl. Data Eng.*. 2006;18(6):770–783. doi: 10.1109/TKDE.2006.94
37. Safavi T, Koutra D. CoDEx: A Comprehensive Knowledge Graph Completion Benchmark. In: Webber B, Cohn T, He Y, Liu Y., eds. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*Association for Computational Linguistics 2020:8328–8350
38. Tanon TP, Weikum G, Suchanek FM. YAGO 4: A Reason-able Knowledge Base. In: . 12123 of *Lecture Notes in Computer Science*. Springer 2020:583–596
39. Lehmann J, Isele R, Jakob M, et al. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web.* 2015;6(2):167–195. doi: 10.3233/SW-140134
40. Singh SK, Kumar S, Mehra PS. Chat GPT & Google Bard AI: A Review. In: 2023:1-6
41. Wei J, Bosma M, Zhao VY, et al. Finetuned Language Models are Zero-Shot Learners. In: OpenReview.net 2022.
42. Touvron H, Lavril T, Izacard G, et al. LLaMA: Open and Efficient Foundation Language Models. *CoRR.* 2023;abs/2302.13971. doi: 10.48550/ARXIV.2302.13971
43. Kandpal N, Deng H, Roberts A, Wallace E, Raffel C. Large Language Models Struggle to Learn Long-Tail Knowledge. In: Krause A, Brunskill E, Cho K, Engelhardt B, Sabato S, Scarlett J., eds. *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*. 202 of *Proceedings of Machine Learning Research*. PMLR 2023:15696–15707.
44. Deng C, Zhang T, He Z, et al. K2: A Foundation Language Model for Geoscience Knowledge Understanding and Utilization. In: WSDM '24. Association for Computing Machinery 2024; New York, NY, USA:161–170
45. Bang Y, Cahyawijaya S, Lee N, et al. A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity. *CoRR.* 2023;abs/2302.04023. doi: 10.48550/arXiv.2302.04023
46. Luo L, Ju J, Xiong B, Li Y, Haffari G, Pan S. ChatRule: Mining Logical Rules with Large Language Models for Knowledge Graph Reasoning. *CoRR.* 2023;abs/2309.01538. doi: 10.48550/ARXIV.2309.01538
47. Manegold S, Boncz PA, Kersten ML. Optimizing Main-Memory Join on Modern Hardware. *IEEE Trans. Knowl. Data Eng.*. 2002;14(4):709–730. doi: 10.1109/TKDE.2002.1019210
48. Peng H, Yang R, Wang Z, et al. Lime: Low-Cost and Incremental Learning for Dynamic Heterogeneous Information Networks. *IEEE Trans. Computers.* 2022;71(3):628–642. doi: 10.1109/TC.2021.3057082
49. Amato D, Bosco GL, Giancarlo R. Standard versus uniform binary search and their variants in learned static indexing: The case of the searching on sorted data benchmarking software platform. *Softw. Pract. Exp.*. 2023;53(2):318–346. doi: 10.1002/SPE.3150
50. Krishnaraj N, Elhoseny M, Lydia EL, Shankar K, Aldabbas O. An efficient radix trie-based semantic visual indexing model for large-scale image retrieval in cloud environment. *Softw. Pract. Exp.*. 2021;51(3):489–502. doi: 10.1002/SPE.2834
51. Meilicke C, Chekol MW, Betz P, Fink M, Stuckenschmidt H. Anytime bottom-up rule learning for large-scale knowledge graph completion. *VLDB J.*. 2024;33(1):131–161. doi: 10.1007/S00778-023-00800-5
52. Cheng K, Ahmed NK, Sun Y. Neural Compositional Rule Learning for Knowledge Graph Reasoning. In: OpenReview.net 2023.
53. Xu Z, Ye P, Chen H, Zhao M, Chen H, Zhang W. Ruleformer: Context-aware Rule Mining over Knowledge Graph. In: Calzolari N, Huang C, Kim H, et al., eds. *Proceedings of the 29th International Conference on Computational Linguistics, COLING 2022, Gyeongju, Republic of Korea,*

*October 12-17, 2022*International Committee on Computational Linguistics 2022:2551–2560.