Inferring Restricted Regular Expressions with Interleaving from Positive and Negative Samples

Yeting Li[1,2], Lingqi Zhang[3], Haiming Chen[1], Bo Huang[4], Jianzhao Zhang[1,2]

[1]University of Chinese Academy of Sciences, Beijing, China

[2]Institute of Software, Chinese Academy of Sciences, Beijing, China

[3]Beijing University of Technology, Beijing, China

[4]Northwestern Polytechnical University, Xi'an, China

Introduction

Learning Regular Expressions with Interleaving

A classical problem in grammatical inference is to identify a language from examples.

We study learning regular expressions (REs) with interleaving (shuffle), denoted by RE(&).

Studying the inference of RE(&) has several practical motivations, such as schema inference.

Introduction

Learning Regular Expressions with Interleaving

Previously, RE(&) learning has been studied from positive examples. However, negative examples might be useful in some applications. For instance, the schema evolution can be done incrementally, with little feedback needed from the user, when we also allow negative examples.

Introduction

Learning Regular Expressions with Interleaving

Learning RE(&) from positive and negative examples may have other crucial applications, such as mining scientific workflows. REs have already been used in the literature as a well-suited mechanism for inter-workflow coordination. The user labeled some sequences of modules from a set of available workflows as positive or negative examples. So such algorithms can be thus applied to infer the workflow pattern that the user has in mind.

Introduction

Learning Regular Expressions with Interleaving

Existing work on RE(&) learning are all working on specific subclasses of REs. The aim of these approaches is to infer restricted subclasses of single occurrence REs with interleaving starting from a positive set of words based on maximum clique or maximum independent set.

Introduction

Learning Regular Expressions with Interleaving

We focus on learning the subclass of RE(&), called SIREs (see Definition 1). Here, we solve this problem by using genetic algorithms. As a result, when given both positive and negative examples, we can effectively learn a SIRE.

Preliminaries

Regular Expression with Interleaving. Let Σ be a finite alphabet of symbols. The set of all words over Σ is denoted by Σ^*. The empty word is denoted by ε. A RE with interleaving over Σ is defined inductively as follows: ε or $a \in \Sigma$ is a RE, for REs r_1 and r_2, the disjunction $r_1|r_2$, the concatenation $r_1 \cdot r_2$, the interleaving $r_1 \& r_2$, or the Kleene-Star r_1^* is also a RE. $r^?$ and r^+ are abbreviations of $r|\varepsilon$ and $r \cdot r^*$, respectively. They are denoted as RE(&).

For example, L(ab&cd) = {cdab, cadb, cabd, acdb, acbd, abcd}

Preliminaries

SIREs

A RE with interleaving r is **SOIRE**, if every alphabet symbol occurs at most once in r. We consider the subclass of REs with interleaving (SIREs) defined by the following grammar.

Definition 1. *The subclass of REs with interleaving (SIREs) are SOIREs over Σ defined by the following grammar:*

$$S ::= T\&S|T$$
$$T ::= \varepsilon|a|a^*|TT, \text{where } a \in \Sigma$$

For instance, $a^*b^?\&cd^+$ is a SIRE, but $a^+b\&c^+a$ is not because a appears twice.

Preliminaries

Definition 2 ***Candidate Region (CR).*** *We use candidate region to define the skeleton structure of a SIRE. Let* $\mathbb{N} = \{0,1,2,\ldots\}$, $\mathbb{N}_0 = \mathbb{N}\backslash\{0\}$ *(0 is excluded). For a SIRE* $r := D_1\& \cdots \& D_n$ *where* $D_i \in \Sigma^*$, $1 \leq i \leq n$, $1 \leq n \leq \mathbb{N}_0$, *it belongs to the candidate region* $|D_1|\& \cdots \& |D_n|$. *The size of* D_i, *denoted by* $|D_i|$, *is the total number of alphabet symbols occurred in* D_i.

Preliminaries

For a given alphabet $|\Sigma| = n$, it is easy to see there are 2^{n-1} CRs. For example, consider Σ=$\{a, b, c, d, e\}$ and $|\Sigma|$=5. As is shown in Fig. 1, we can get 16 CRs. The number of squares with the same color represents the $|D_i|$, e.g., the 6th CR denotes 1&1&3 and the 12th CR denotes 1&1&1&2. So, the SIRE $r_1 = a^+\&b\&c^*d^+e^?$ belongs to the 6th CR 1&1&3 and the SIRE $r_2 = a^+\&b\&c^*\&d^+e^?$ belongs to the 12th CR 1&1&1&2.

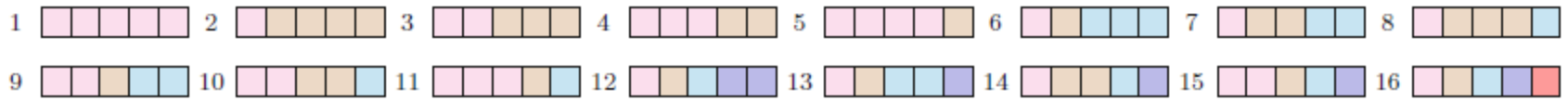

Fig. 1. All the candidate regions of $|\Sigma| = 5$

Preliminaries

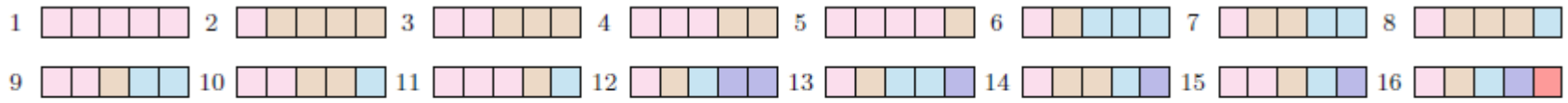

Fig. 1. All the candidate regions of $|\Sigma| = 5$

From Definition 2, when $|\Sigma| = n$, there are 2^{n-1} CRs. Because of the unorder features of SIREs, we can easily find that for a SIRE $r = D_1 \& \cdots \& D_n$, the order of D_i can be arbitrary, where $1 \leq i \leq n$. Hence, we can merge some equivalent CRs and get the SCRs. For instance, in Fig. 1, we can merge the 6th CR 1&1&3, the 8th CR 1&3&1 and the 11th CR 3&1&1 together. After the merger of some equivalent CRs, we get the SCRs shown in Fig. 2.

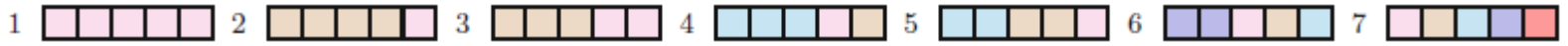

Fig. 2. All the simplified candidate regions of $|\Sigma| = 5$

Preliminaries

When the $|\Sigma| = n$*, how many SCRs are there?* This problem is equivalent to Integer Partition, e.g., when $|\Sigma| = 5$, there are 7 SCRs, including 5, 4&1, 3&2, 3&1&1, 2&2&1, 2&1&1&1 and 1&1&1&1&1. Meanwhile, the 7 partitions of 5 are: $5=5$, $5=4+1$, $5=3+2$, $5=3+1+1$, $5=2+2+1$, $5=2+1+1+1$ and $5=1+1+1+1+1$. In general, approximation formulas exist that can calculate the number of partitions. For $n \in \mathbb{N}$, the number of partitions of n $p(n) \approx \frac{1}{4n\sqrt{3}} e^{\pi\sqrt{\frac{2n}{3}}}$ as $n \to \infty$. As n increases, $\frac{1}{4n\sqrt{3}} e^{\pi\sqrt{\frac{2n}{3}}}$ is far less than 2^{n-1}. It can be seen from Table 1 intuitively that the number of SCRs is far less than CRs. So we use function *getSCRs()* to get SCRs instead of CRs in our algorithm *i*SIRE.

Table 1. The number of CRs and SCRs of varying alphabet size.

$\|\Sigma\|$	5	10	15	20	25	30	35	40
CRs	16	512	16384	524288	16777216	536870912	17179869184	549755813888
SCRs	7	42	176	627	1958	5604	14883	37338

Learning Algorithm

Learning SIREs

Our algorithm aims to obtain an accurate and precise SIRE, which should accept as many positive samples as possible and reject as many negative samples as possible.

Learning Algorithm

Learning SIREs

The algorithm iSIRE first figures out the SCRs of the expression to be learned, then for each SCR, employs genetic algorithms to learn character sequence and multiplicity sequence in parallel, and decodes each learned sequence to a SIRE according to its SCR. After multi-generation evolution and iteration, the best SIRE is selected by function bestRE().

Algorithm 1: *iSIRE*

Input: positive examples S_+, negative examples S_-
Output: a SIRE r

initialize candidate set $C \leftarrow \varnothing$
$\Sigma \leftarrow \text{getAlphabet}(S_+, S_-)$
$SCRs \leftarrow \text{getSCRs}(|\Sigma|)$
foreach $scr \in SCRs$ **in parallel do**
    add $\text{candSIRE}(S_+, S_-, scr)$ to C
return $r \leftarrow \text{bestRE}(C)$

Learning Algorithm

Learning SIREs

Function *bestRE()* is designed to select the best SIRE. It measures two metrics of SIREs: $K(r)$ for *accuracy* and $CC(r)$ [23] for *preciseness*. For a SIRE r, $K(r) = \frac{|T_P|+|T_N|-|F_P|-|F_N|}{|S_+|+|S_-|}$, $T_P = \{w \in S_+ | w \in L(r)\}$, $T_N = \{w \in S_- | w \notin L(r)\}$, $F_P = \{w \in S_+ | w \notin L(r)\}$, $F_N = \{w \in S_- | w \in L(r)\}$, S_+ is the set of positive examples and S_- is the set of negative examples. The Combinatorial Cardinality ($CC(r)$, introduced in [23]) of r can be computed as follows: $CC(r) = \prod_{i=1}^{n-1} \binom{\sum_{j=1}^{i+1} |D_j|}{|D_{i+1}|}$. Note that $K(r)$ has higher priority than $CC(r)$ when selecting the best SIRE. If the value of $K(r)$ is larger, then it means r can accept more positive examples and reject more negative examples. Smaller the $CC(r)$ is, more precise the SIRE will be.

Learning Algorithm

Learning SIREs

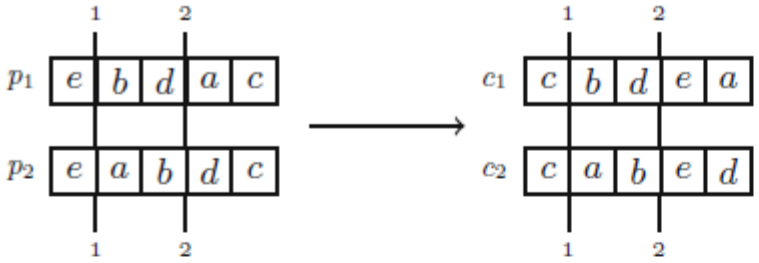

Fig. 3. Character crossover

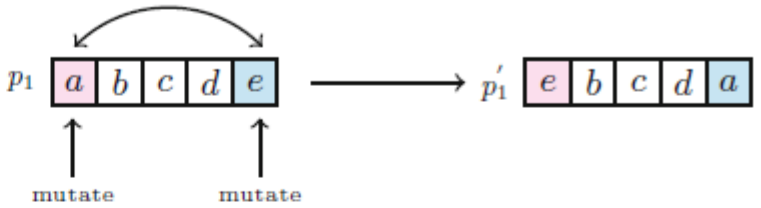

Fig. 4. Character mutation

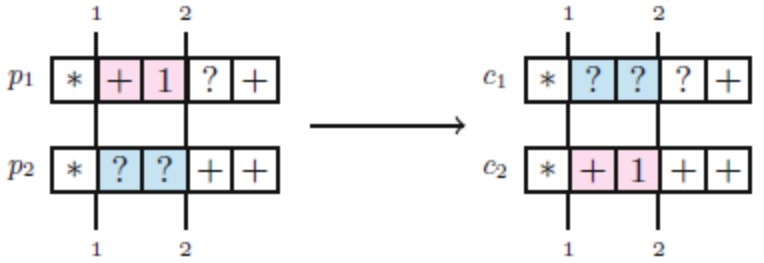

Fig. 7. Multiplicity crossover

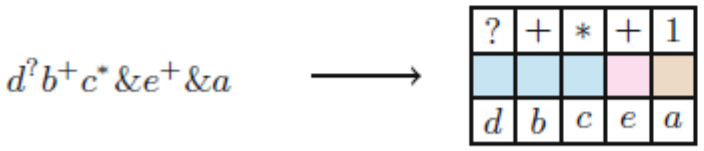

Fig. 5. Chromosome encoding

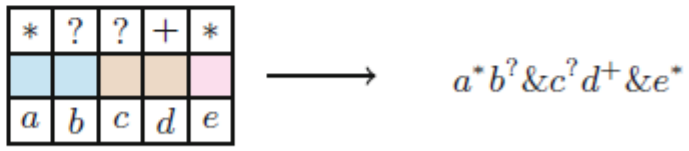

Fig. 6. Chromosome decoding

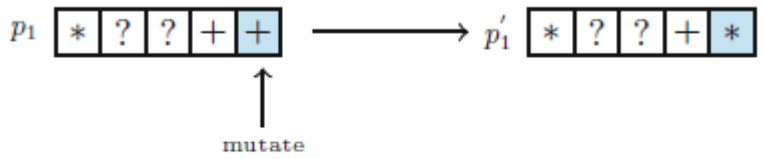

Fig. 8. Multiplicity mutation

Learning Algorithm

A Case Study

Given positive examples $S_+ = \{abcd, aadbc, bdd\}$ and negative examples $S_- = \{dabc, bdca\}$. Because $|\Sigma| = 4$, we can divide 5 SCRs (shown in Table 3) by calling $getSCRs(|\Sigma|)$. As is shown in Table 3, the $K(r)$ values of candidate SIREs in the 2 SCRs are 1. Furthermore, the candidate SIRE in the SCR 2&2 has the minimum $CC(r)$ value. So the inferred SIRE by iSIRE is: $r_2 = a^*d^+\&bc^?$. Note that $S_+ \subseteq L(r_2)$ and $S_- \cap L(r_2) = \varnothing$, namely, $L(r_2)$ accepts all the positive examples and rejects all the negative examples.

Table 3: Learning SIREs from $S_+ = \{abcd, aadbc, bdd\}$ and $S_-=\{dabc, bdca\}$.

SCR	SIRE	$K(r)$	$CC(r)$
4	$a^?bc^?d^+$	0.6	1
3&1	$bc^?d^+\&a^?$	0.6	4
2&2	$\mathbf{a^*d^+\&bc^?}$	**1**	**6**
2&1&1	$a^*d^+\&b\&c^?$	1	12
1&1&1&1	$a^*\&b\&c^?\&d^+$	0.2	24

Experiments

Learning SIREs from Positive Examples

The results learned by iSIRE is **more precise** than the other 3 algorithms (Exact Minimal, conMiner, and conDAG).

Table 2. Result of learning SIREs from positive examples.

$\|\Sigma\|$	$\|S\|$	Exact minimal		conMiner		conDAG		*i*SIRE	
		$K(r)$	$CC(r)$	$K(r)$	$CC(r)$	$K(r)$	$CC(r)$	$K(r)$	$CC(r)$
5	100	100%	20	100%	20	100%	20	100%	20
	500	100%	30	100%	30	100%	30	100%	30
	1000	100%	60	100%	60	100%	60	100%	60
10	100	100%	840	100%	840	100%	840	100%	840
	500	100%	37800	100%	50400	100%	50400	100%	**16800**
	1000	100%	5040	100%	6300	100%	5040	100%	**3150**
15	100	100%	1801800	100%	2522520	100%	2162160	100%	**1261260**
	500	100%	8108100	100%	15135120	100%	10810800	100%	**7207200**
	1000	100%	300300	100%	360360	100%	900900	100%	**270270**

Experiments

Learning SIREs from Positive and Negative Examples

It is easy to observe that majority of SIREs learned by iSIRE **accept most of positive examples and reject most of negative examples**, which demonstrates **the high effectiveness of our algorithms**.

Table 3. Results of learning SIREs from positive and negative examples.

$\|\Sigma\|$	$\|S_+\|$	$\|S_-\|$	$K(r)$	$CC(r)$	$\|\Sigma\|$	$\|S_+\|$	$\|S_-\|$	$K(r)$	$CC(r)$	$\|\Sigma\|$	$\|S_+\|$	$\|S_-\|$	$K(r)$	$CC(r)$
5	25	75	92%	10	10	25	75	82%	360	15	25	75	82%	756756
	50	50	74%	30		50	50	72%	840		50	50	74%	360360
	75	25	90%	5		75	25	78%	7560		75	25	82%	25225200
	300	200	87.6%	10		300	200	82.4%	5040		300	200	82.4%	300300
	250	250	88.8%	20		250	250	81.2%	2520		250	250	70.4%	50450400
	200	300	81.2%	60		200	300	83.2%	4200		200	300	75.6%	900900
	750	250	69.8%	20		750	250	85.6%	75600		750	250	72.2%	37837800
	500	500	84%	30		500	500	91.6%	30240		500	500	73.4%	4054050
	250	750	77.6%	60		250	750	84.8%	25200		250	750	82.6%	378378000

Conclusions

In this paper, we provided algorithm iSIRE to learn a SIRE from **positive and negative examples based on genetic algorithms**.
Then we conducted experiments with alphabets of different sizes, and results showed that with only positive examples, our learning results are **more precise** compared with the state-of-the-art algorithms, and when given both positive and negative examples, we can learn SIREs with **high accuracy**.

Thanks for attention!