On Representations of Ternary Order Relations in Numeric Strings

Jinil Kim[1,*], Amihood Amir[2,3], Joong Chae Na[4], Kunsoo Park[1,*],

Jeong Seop Sim[5]

[1]Dep. of Computer Science and Engineering, Seoul National University, Korea
[2]Department of Computer Science, Bar-Ilan University, Israel
[3]Johns Hopkins University, USA
[4]Dep. of Computer Science and Engineering, Sejong University, Korea
[5]Dep. of Computer and Information Engineering, Inha University, Korea
[*]{jikim|kpark}@theory.snu.ac.kr

Abstract

Order-preserving matching is a string matching problem of two numeric strings where the relative orders of consecutive substrings are matched instead of the characters themselves. The order relation between two characters is a ternary relation $(>, <, =)$ rather than a binary relation $(>, <)$, but it was not sufficiently studied in previous works [5, 7, 1]. In this paper, we extend the representations of order relations by Kim et al. [5] to ternary order relations, and prove the equivalence of those representations. The *extended prefix representation* takes $\log m + 1$ bits per character, while the *nearest neighbor representation* takes $2 \log m$ bits per character. With our extensions, the time complexities of order-preserving matching in binary order relations can be achieved in ternary order relations as well.

1 Introduction

Order-preserving matching is a string matching problem of two numeric strings where the relative orders of substrings are matched instead of the characters themselves. It has many practical applications such as stock price analysis and musical melody matching. The study on this field was introduced by Kubica et al. [7] and Kim et al. [5] where Kubica et al. [7] defined order relations by order isomorphism of two strings, while Kim et al. [5] defined them explicitly by the sequence

In: Costas S. Iliopoulos, Alessio Langiu (eds.): Proceedings of the 2nd International Conference on Algorithms for Big Data, Palermo, Italy, 7-9 April 2014, published at http://ceur-ws.org/

of rank values (which they called the *natural representation*). Recently, variants of order-preserving matching have been studied such as order-preserving suffix trees [2], approximate matching with k mismatches [3], and a Boyer-Moore type algorithm [1].

An order relation between two characters is a ternary relation $(>, <, =)$ rather than a binary relation $(>, <)$, but the representations of order relations were not sufficiently studied for ternary order relations. Kim et al. [5] considered only binary order relations by assuming that all the characters in a string are distinct, while Kubica et al. [7] considered ternary order relations but their match condition on equal characters was faulty and it was fixed later by Cho et al. [1]. As there was no integrated study on the relationship between the representations of order relations, previous works [1, 2, 3] individually handled the cumbersome details of ternary order relations on their own works.

The number of order relations on a sequence of n elements is $n!$ in binary order relations, but it is the *ordered Bell number* [4] in ternary order relations. The ordered Bell number is the solution of the recurrence $f(n) = 1 + \sum_{j=1}^{n-1} \binom{n}{j} f(n-j)$, which is exponentially greater than $n!$. The representations of order relations should be extended for ternary order relations, which might incur some space overhead on the representations.

In this paper, we extend the representations of order relations by Kim et al. [5] to ternary order relations, and prove the equivalence of those representations. With the *extended prefix representation*, order-preserving matching can be done in $O(n \log m)$ time, and the representation of order relations takes $(\log m + 1)$-bit per character. With the *nearest neighbor representation*, the matching can be done in $O(n + m \log m)$, but the representation takes $(2 \log m)$-bit per character. The *nearest neighbor representation* is suitable for the single pattern matching while the *extended prefix representation* is space-efficient and can be useful in order-preserving multiple pattern matching [5] and order-preserving suffix trees [2].

2 Problem Formulation

Let Σ denote the set of numbers such that a comparison of two numbers can be done in constant time, and let Σ^* denote the set of strings over the alphabet Σ. For a string $x \in \Sigma^*$, let $|x|$ denote the length of x. A string x is described by a sequence of characters $(x[1], x[2], ..., x[|x|])$. Let a substring $x[i..j]$ be $(x[i], x[i+1], ..., x[j])$ and a prefix x_i be $x[1..i]$. For a character $c \in \Sigma$, let $rank_x(c) = 1 + |\{i : x[i] < c \text{ for } 1 \leq i \leq |x|\}|$, and let $exist_x(c)$ be 1 if c exists in x, and 0 otherwise. Let $ex\text{-}rank_x(c) = (rank_x(c), exist_x(c))$. For any boolean condition *cond*, let $\delta(cond)$ be 1 if *cond* is true, 0 otherwise.

Order Isomorphism [7] For two strings x and y of length n, x and y are order-isomorphic if $\forall i, j \in [1..n]$, $x[i] \leq x[j] \Leftrightarrow y[i] \leq y[j]$.

Order isomorphism *implicitly* deals with ternary order relations because each of ternary order relations $(>, <, =)$ can be checked by variants of the proposition in Definition 2 (changing i and j, taking the contrapositive, or both). For example,

if $x[i] > x[j]$, then $y[i] > y[j]$ by the contrapositive of $x[i] \leq x[j] \Leftarrow y[i] \leq y[j]$. If $x[i] = x[j]$, then $y[i] = y[j]$ by $x[i] \leq x[j] \Rightarrow y[i] \leq y[j]$ and $x[j] \leq x[i] \Rightarrow y[j] \leq y[i]$.

The definition of order isomorphism looks simple, but it is somewhat complicated to handle in practice. The number of the order relations involved in checking order isomorphism of two strings of length n is $O(n^2)$, hence it has an inherent quadratic term if the definition is used directly for order-preserving matching.

Natural Representation [5] For a string x of length n, the natural representation of the order relations is defined as $Nat(x) = (rank_x(x[1]), rank_x(x[2]), ..., rank_x(x[n]))$.

In the natural representation, ternary order relations are *explicitly* stated in terms of ranks. For example, $x[i] > x[j]$ if and only if $Nat(x)[i] > Nat(x)[j]$, and $x[i] = x[j]$ if and only if $Nat(x)[i] = Nat(x)[j]$. That is, the order relations of two strings coincide if and only if $Nat(x) = Nat(y)$. The comparison of two natural representations takes $O(n)$ time if the natural representations are given.

These two definitions are equivalent because the natural representations of two strings are identical if and only if they are order-isomorphic. We adopt the natural representation throughout this paper because the definition itself and subsequent analysis are more intuitive.

Order-Preserving Matching [5] Given a text $T[1..n] \in \Sigma^*$ and a pattern $P[1..m] \in \Sigma^*$, P matches T at position i if $Nat(P) = Nat(T[i-m+1..i])$. Order-preserving matching is the problem of finding all positions of T matched with P.

Equivalent Representation For any two representations $R_1(\cdot)$ and $R_2(\cdot)$, R_1 is equivalent to R_2 if $R_1(x) = R_1(y) \Leftrightarrow R_2(x) = R_2(y)$ for any two strings x, y.

For KMP-based algorithms [6], the length of matches is incrementally increased when the next character of the text matches that of the pattern. Such a match operation is formalized by the *match condition* as follows.

Match Condition A match condition of a representation $R(\cdot)$ is a boolean function $Match(x, y, R(x), t+1)$ such that $Nat(x_{t+1}) = Nat(y_{t+1})$ holds if and only if $Nat(x_t) = Nat(y_t)$ and $Match(x, y, R(x), t+1)$ where x, $y \in \Sigma^*$, $|x| = |y|$ and $t \in [1..|x|-1]$.

3 Prefix Representation

Prefix Representation [5] For a string x, the prefix representation of the order relations is defined as $Pre(x) = (rank_{x_1}(x[1]), rank_{x_2}(x[2]), ..., rank_{x_{|x|}}(x[|x|]))$.

The *prefix representation* has an ambiguity between different strings in ternary order relations [1]. For example, when $x = (10, 30, 20)$, and $y = (10, 20, 20)$, the prefix representations of both x and y are $(1, 2, 2)$. The ambiguity is resolved in the extended prefix representation.

i	1	2	3	4	5	6	7	8
x	30	10	50	20	30	20	25	20
$Nat(x_6)$	4	1	6	2	4	2		
Ex-pre(x_7)	$\binom{1}{0}$	$\binom{1}{0}$	$\binom{3}{0}$	$\binom{2}{0}$	$\binom{3}{1}$	$\binom{2}{1}$	$\binom{4}{0}$	
$NN(x_7)$	$\binom{-\infty}{\infty}$	$\binom{-\infty}{1}$	$\binom{2}{1}$	$\binom{2}{1}$	$\binom{1}{1}$	$\binom{4}{4}$	$\binom{6}{5}$	
$Nat(x_7)$	5	1	7	2	5	2	4	
Ex-pre(x_8)	$\binom{1}{0}$	$\binom{1}{0}$	$\binom{3}{0}$	$\binom{2}{0}$	$\binom{3}{1}$	$\binom{2}{1}$	$\binom{4}{0}$	$\binom{2}{1}$
$NN(x_8)$	$\binom{-\infty}{\infty}$	$\binom{-\infty}{1}$	$\binom{2}{1}$	$\binom{2}{1}$	$\binom{1}{1}$	$\binom{4}{4}$	$\binom{6}{5}$	$\binom{6}{6}$
$Nat(x_8)$	6	1	8	2	6	2	5	2

Figure 1: Example of Lemma 3.1 and Lemma 4.1

Extended Prefix Representation For a string x, the extended prefix representation is defined as *Ex-pre*(x) = (*ex-rank*$_{x_1}(x[1])$, *ex-rank*$(x_2[2])$, ..., *ex-rank*$(x_{|x|}[|x|])$).

For any $t \in [1..|x|-1]$, $Nat(x_{t+1})$ can be computed from $Nat(x_t)$ and *Ex-pre*(x_{t+1}).

Lemma 3.1 (Representation Conversion) *Given* $Nat(x_t)$ *and* *Ex-pre*(x_{t+1}),

$$Nat(x_{t+1})[i] = \begin{cases} a + \delta((a > c) \vee (a = c \wedge d = 0)) & \text{for } 1 \le i \le t \\ c & \text{for } i = t+1 \end{cases}$$

where $a = Nat(x_t)[i]$ *and* $\binom{c}{d} =$ *Ex-pre*$(x_{t+1})[t+1]$.

An example of Lemma 3.1 is shown in Figure 1 for $x = (30, 10, 50, 20, 30, 20, 25, 20)$. Let's consider when $t+1 = 7$. For $i = 1$, we have $Nat(x_6)[1] = a = 4$ and *Ex-pre*$(x_7)[7] = \binom{c}{d} = \binom{4}{0}$. Since $a = c$ and $d = 0$, we have $Nat(x_7)[1] = a + 1 = 5$. For $i = 2$, $\binom{c}{d}$ is the same as for $i = 1$, and we have $Nat(x_6)[2] = a = 1$. Since $a < c$, $Nat(x_7)[2] = a = 1$. Let's consider the next step when $t+1 = 8$. For $i = 4$, we have $Nat(x_7)[4] = a = 2$ and $\binom{c}{d} = \binom{2}{1}$. As $a = c$ and $d = 1$, $Nat(x_8)[4] = a = 2$.

Theorem 3.2 *Ex-pre*$(\cdot)$ *is equivalent to* $Nat(\cdot)$.

Theorem 3.3 (Match Condition) *Given* x, y *and* t, *the condition* *Ex-pre*$(x_{t+1})[t+1]$ = *Ex-pre*$(y_{t+1})[t+1]$ *is a match condition of* *Ex-pre*$(\cdot)$.

4 Nearest Neighbor Representation

Let $LMax_x[i] = j$ if $x[j] = \max\{x[k] : x[k] \le x[i]$ for $1 \le k \le i-1\}$, or $-\infty$ if no such j. Let $LMin_x[i] = j$ if $x[j] = \min\{x[k] : x[k] \ge x[i]$ for $1 \le k \le i-1\}$, or

∞ if no such j. If there are multiple j's for $LMax_x[i]$ or $LMin_x[i]$, we choose the rightmost one. In Figure 1, $LMax_x[8] = 6$ since $x[6]$ is the rightmost one among the maximum values which are less than or equal to $x[8]$ in $x[1..7]$. Similarly, $LMin_x[8] = 6$.

Nearest Neighbor Representation [5, 7, 1] For a string x, the nearest neighbor representation can be defined as

$$NN(x) = \binom{LMax_x[1]}{LMin_x[1]}\binom{LMax_x[2]}{LMin_x[2]}\cdots\binom{LMax_x[|x|]}{LMin_x[|x|]}$$

For convenience, let $x[-\infty] = -\infty$, $x[\infty] = \infty$, $Nat(x)[-\infty] = 0$ and $Nat(x)[\infty] = |x| + 1$ for any string x. Then, $Nat(x)[LMax_x[i]] \le Nat(x)[i] \le Nat(x)[LMin_x[i]]$ holds for any $i \in [1..|x|]$ even when $LMax_x[i] = -\infty$ or $LMin_x[i] = \infty$.

Lemma 4.1 (Representation Conversion) *Given $Nat(x_t)$ and $NN(x_{t+1})$,*

$$Nat(x_{t+1})[i] = \begin{cases} a + \delta((a > f) \vee (a = f \wedge e \neq f)) & \text{for } 1 \le i \le t \\ f & \text{for } i = t + 1 \end{cases}$$

where $a = Nat(x_t)[i]$, $\binom{c}{d} = NN(x_{t+1})[t+1]$, $e = Nat(x_t)[c]$ and $f = Nat(x_t)[d]$.

An example of Lemma 4.1 is shown in Figure 1 for $x = (30, 10, 50, 20, 30, 20, 25, 20)$. Let us consider when $t + 1 = 7$. For $i = 1$, we have $Nat(x_6)[1] = a = 4$, $NN(x_7)[7] = \binom{c}{d} = \binom{6}{5}$, $e = 2$ and $f = 4$. Since $a = f$ and $e \neq f$, we get $Nat(x_7)[1] = a + 1 = 5$. For $i = 2$, $\binom{c}{d}$, e and f are the same as for $i = 1$, and we have $a = 1$. Since $a < e$, we have $Nat(x_7)[2] = a = 1$. For $i = 3$, we have $a = 6$ and $a > f$, and thus $Nat(x_7)[3] = a + 1 = 7$. For $i = 4$, we have $a = 2$. Since $a = e$ and $e \neq f$, we get $Nat(x_7)[4] = a = 2$. For $i = 7$, we get $Nat(x_7)[7] = f = 4$ since $i = t + 1$. Consider the next step when $t + 1 = 8$. For $i = 4$, we have $a = 2$, $NN(x_8)[8] = \binom{c}{d} = \binom{2}{1}$, $e = 2$ and $f = 2$. Since $a = e = f$, we get $Nat(x_8)[4] = a = 2$.

Theorem 4.2 *$NN(\cdot)$ is equivalent to $Nat(\cdot)$.*

A naive match condition of the nearest neighbor representation is $NN(x_{t+1})[t+1] = NN(y_{t+1})[t+1]$ as that of the extended prefix representation in Theorem 3.3, which requires computing the nearest neighbor representations of both x and y. Kubica et al. [7] proposed an efficient match condition for ternary order relations which can be checked in constant time when the nearest neighbor representation of x is given, but it was faulty. Cho et al. [1] presented a modified match condition in ternary order relations, which can produce an $O(n + m \log m)$ algorithm as in binary order relations.

Theorem 4.3 (Match Condition of Nearest Neighbor Representation [1]) *Given x, y and t, the condition $(y[c] < y[t+1] < y[d]) \vee (y[t+1] = y[c] = y[d])$ is a match condition of $NN(\cdot)$ where $\binom{c}{d} = NN(x_{t+1})[t+1]$.*

We can generalize order-preserving matching algorithms in binary order relations [5] to ternary order relations using the representations above. For single pattern matching, we can obtain an $O(n \log m)$ algorithm with the extended prefix representation, and an $O(n + m \log m)$ algorithm with the nearest neighbor representation, both of which are consistent with the results in [5, 7, 1]. For multiple pattern matching, we can obtain an $O((n + m) \log m)$ algorithm in ternary order relations using the extended prefix representation.

Acknowledgements

The work of Jinil Kim and Kunsoo Park was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2011-0029924). Amihood Amir was partly supported by NSF grant CCR-09-04581, ISF grant 347/09, and BSF grant 2008217. Joong Chae Na was supported by the IT R&D program of MKE/KEIT [10038768, The Development of Supercomputing System for the Genome Analysis], and by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2011-0007860). Jeong Seop Sim was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2012R1A2A2A01014892), and by the Industrial Strategic technology development program (10041971, Development of Power Efficient High-Performance Multimedia Contents Service Technology using Context-Adapting Distributed Transcoding) funded by the Ministry of Knowledge Economy (MKE, Korea).

References

[1] S. Cho, J. C. Na, K. Park, and J. S. Sim. Fast order-preserving pattern matching. In *COCOA*, 2013.

[2] M. Crochemore, C. S. Iliopoulos, T. Kociumaka, M. Kubica, A. Langiu, S. P. Pissis, J. Radoszewski, W. Rytter, and T. Walen. Order-preserving incomplete suffix trees and order-preserving indexes. In *SPIRE*, pages 84–95, 2013.

[3] P. Gawrychowski and P. Uznanski. Order-preserving pattern matching with k mismatches. *CoRR*, abs/1309.6453, 2013.

[4] O. A. Gross. Preferential arrangements. *The American Mathematical Monthly*, 69:4–8, 1962.

[5] J. Kim, P. Eades, R. Fleischer, S.-H. Hong, C. S. Iliopoulos, K. Park, S. J. Puglisi, and T. Tokuyama. Order-preserving matching. *To appear in Theor. Comput. Sci.*

[6] D. E. Knuth, J. H. M. Jr., and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

[7] M. Kubica, T. Kulczynski, J. Radoszewski, W. Rytter, and T. Walen. A linear time algorithm for consecutive permutation pattern matching. *Inf. Process. Lett.*, 113(12):430–433, 2013.