# A Nettree for Pattern Matching with Flexible Wildcard Constraints*

Youxi Wu[1,2], Xindong Wu[3,2], Fan Min[4], Yan Li[5]
*1. School of Computer Science and Software, Hebei University of Technology, Tianjin 300130, China*
*2. Department of Computer Science, University of Vermont, Burlington, VT 05405, USA*
*3. School of Computer Science & Information Engineering, Hefei University of Technology, Hefei 230009, China*
*4. School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China*
*5. School of Management, Hebei University of Technology, Tianjin 300130, China*
*{youxiwu, xwu}@cems.uvm.edu; minfan@uestc.edu.cn; lywuc@163.com*

## Abstract

*In this paper, a new nonlinear structure called Nettree is proposed. A Nettree is different from a tree in that a node may have more than one parent. An algorithm, named Nettree for pAttern Matching with flExible wIldcard Constraints (NAMEIC), based on Nettree is designed to solve pattern matching with flexible wildcard constraints. The problem is exponential with regard to the pattern length m. We prove the correctness of the algorithm, and illustrate how it works through an example. NAMEIC is W*m times faster than an existing approach because the result can be given after creating the Nettree in one pass, where W is the maximal gap flexibility. Experiments validate the correctness and efficiency of NAMEIC.*

**Keywords:** Nettree, Pattern matching, Flexible wildcard

## 1. Introduction

Many real-problems such as biological sequence analysis [1,2], text indexing [3,4], time series data mining [5], stream data mining [6,7] and so on involve pattern matching (or string matching or text matching) with wildcards ("don't cares"), often marked as "*" (or "?", "$\phi$", "#") which can match any letter in a given set of symbols [3,8, 9]. There are mainly two kinds of research efforts for the problem of string matching with wildcards. The first one is engaged to a fixed length of wildcards. Fischer et al. [10] developed an algorithm for solving pattern matching with wildcards, in which the number of wildcards between two consecutive letters in $P$ is a constant. Cole et al. [3] concentrated on the total of wildcards which is fixed in $P$. The disadvantage of this kind of research is that the length of wildcards is not a range. But in a general case, it is difficult to know every length of wildcards between every two consecutive letters in pattern $P$ in advance. So the length of wildcards cannot be a constant but a range [11, 12]. Recently, the problem with flexible gap constraints has attracted extensive attention. He et al [2] aimed to solve the problem of frequent pattern mining without user specified gap constraints. Huang et al [13] considered the problem of mining a set of gap constrained sequential patterns across multiple sequences. Zhu et al [14] studied mining frequent patterns with gaps and the one-off condition. Chen et al [11] proposed an algorithm, SAIL, by which the optimal occurrences under the one off condition were found. Min et al [12] introduced an algorithm, PAIG-RST (reduced space and time), by which the number of occurrences of the problem was computed. In [12], not only the length of wildcards between every two consecutive letters (which is called complex local constraints) but also the length of all occurrences (which is called global length constraints) are ranges. But when it is not necessary to consider the global length constraints, PAIG-RST is not highly efficient because some local constraints are recalculated.

In order to avoid recalculating, a more efficient algorithm Nettree for pAttern Matching with flExible wIldcard Constraints (NAMEIC), based on a new nonlinear data structure Nettree, is proposed in this paper. A Nettree is a kind of directed acyclic graph (DAG) with edge labels. The concept, structure and creation rules of Nettree are given. Then the proof of its correctness is provided. An example is used to illustrate how our NAMEIC works. The time complexity of the algorithm is $O(W*m*n)$, where $n$ is the length of $S$, $m$ is the length of $P$ and $W$ is the maximal gap flexibility. NAMEIC is $W*m$

* This research is supported by the National Natural Foundation of China (NSFC) under grant No. 60828005.

times faster than PAIG-RST whose time complexity is $O(W^2*m^2*n)$.

In summary, our contributions in this paper are as follows:

- We present a new nonlinear structure called Nettree. A Nettree is different from a tree in that a node may have more than one parent. To our best knowledge, this is the first study on this nonlinear structure.
- An algorithm (NAMEIC) based on Nettree is proposed to solve pattern matching with flexible wildcard constraints.

The rest of this paper is organized as follows. In Section 2 the definition of the problem is given. In Section 3 the concept and structure of Nettree are explained at first. Then the design of NAMEIC is provided. After this the correctness of the algorithm is proved and an illustration example is used to show how the algorithm works. In Section 4 the time and space complexity of the algorithm and the upper bound of the problem are analyzed. In Section 5 our experiments demonstrate the correctness of our analysis of the problem. We conclude in Section 6.

## 2. Problem Formulation

In this section, we give a brief introduction of the problem first defined in [11].

**Definition 1. Pattern Matching with Flexible Wildcard Constraints (PMFWC).**

Given a **pattern** $P=p_0[\min_0,\max_0]\ p_1 \cdots [\min_{j-1}, \max_{j-1}]\ p_j \cdots\ p_{m-2}[\min_{m-2}, \max_{m-2}]\ p_{m-1}$ and a **sequence** $S=s_0s_1\cdots s_i\cdots s_{n-1}$, where $m$ is the **length of pattern** $P$, $p_j \neq \phi$, $0<j \leq m-1$, $n$ is the **length of sequence** $S$, $s_i \neq \phi$ and $0 \leq i \leq n-1$. $\phi$ is referred to as **wildcard** which denotes a letter and can match any letter in a given alphabet. $\min_{j-1}$ and $\max_{j-1}$ are given integer values and mean the minimal and maximal length of wildcards between two given letters $p_{j-1}$ and $p_j$ respectively, where $0 \leq \min_{j-1} \leq \max_{j-1}$. $p_{j-1}$ and $p_j$ are two **consecutive letters**. When $\min_{j-1}=\max_{j-1}=0$, $p_{j-1}[0,0]p_j$ can be written as $p_{j-1}\ p_j$.

If there exists a sequence $A$ of position indices $\{a_0 \cdots a_j \cdots a_{m-1}\}$ which, subject to the following equation, is an **occurrence** of pattern $P$ in sequence $S$,

$$p_j = s_{a_j}$$
$$\text{subject to } \min_{j-1} \leq a_j - a_{j-1} - 1 \leq \max_{j-1}, \quad (1)$$
$$a_{j-1} < a_j$$

where $0 \leq a_j \leq n-1$.

$a_{m-1} - a_0 + 1$ is referred to as the **length of the occurrence** $A$. $\min_{j-1}$ and $\max_{j-1}$ are **flexible wildcard constraints**.

When pattern $P$ and sequence $S$ are given, the focus of the paper is to compute the number of all occurrences, $N(S,P)$.

## 3. Nettree and Algorithm Design

### 3.1. The Definition of Nettree

**Definition 2.** A Nettree is a kind of DAG with two kinds of edge labels, "parent-child" and "child-parent", where each node has zero or more children nodes and zero or more parents nodes. Furthermore, the children of each node have a specific order. A Nettree has the following three properties.

1. A Nettree is an extension of a tree because it has all concepts of a tree, such as the root, leaf, level, parent, child and so on.

2. A Nettree may have $r$ roots, where $r \geq 1$.

3. Some nodes except roots in a Nettree may have $q$ parents, where $q \geq 1$.

A Nettree is shown in Fig. 1. Nodes A and B are two roots of the Nettree. Nodes C, F and G are three leaves. Node D has two parents (nodes A and B). In Fig.1, each edge has a label either "parent-child" or "child-parent". So it is a kind of DAG with edge labels. Otherwise there will be a cycle {B, D, G, E, B}.

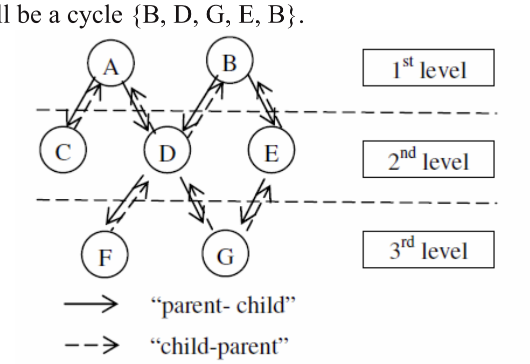


Figure 1. A Nettree.

**Definition 3.** Node $t$ $(0 \leq t \leq n-1)$ in the $j^{th}$ $(1 \leq j \leq m)$ level is denoted by $n_j^t$. $Path(n_j^t)$ denotes the number of **root paths** of node $n_j^t$ (from all first level nodes to node $n_j^t$). The number of root paths of a first level node is 1 i.e., $Path(n_1^t)=1$.

Another characteristic of Nettree is that there may be more than 1 root path from a root to a node in a Nettree. For example, there are 2 root paths from root B to node G ({B, D, G} and {B, E, G}) in Fig. 1.

**Property 1.** The number of root paths of a $j^{th}$ level node ($Path(n_j^t)$) is the sum of root paths of its parents

$$Path(n_j^t) = \sum_{i=1}^{k} Path(n_{j-1}^{t_i}), \quad (2)$$

where $k$ is the number of parents of node $n_j^t$ .

### 3.2. The structure of Nettree

In order to solve the pattern matching problem in Definition 1, the structures of Nettree levels and Nettree nodes are used.

The structure of **Nettree levels** is used to link all the nodes and count the number of nodes of each level. This structure has two fields i.e. data field and pointer field. A data field contains two kinds of data i.e. char and number which represent the common character of nodes and the number of nodes in this level respectively. A pointer field contains three pointers i.e. head pointer, start pointer and tail pointer. The head pointer points to the first node in this level. The start pointer points to the first possible parent of the next level node and the tail pointer points to the last node of this level.

There are six fields in the structure of **Nettree nodes** (shown in Fig. 2). The degrees of parents and children represent the numbers of its parents and its children respectively. The pointer arrays of parents and children contain parents and children of the current node respectively. The next pointer contains a successor of the current node in the same level. A data field contains two numbers which represent the position of the sequence and the number of its root paths respectively.

| | |
|---|---|
| The degree of parents | Pointer array of parents |
| Data field | Next pointer |
| The degree of children | Pointer array of children |

Figure 2. The structure of Nettree nodes.

### 3.3. Proposed algorithm

When a letter $s_i$ ( $0 \le i \le n-1$ ) arrives, we check whether $s_i$ satisfies the following three rules or not. If yes, we create a node or an edge according to the rules.

**Rule 1. Creation of a first level node**.

If $s_i = p_0$, node $n_1^i$ will be created in the first level and the node will be added in the tail of the first level and the number of the first level nodes should be increased by 1.

**Rule 2. Creation of a $j$+1$^{th}$ (j>0) level node**.

If $s_i = p_j$ and the distance between $i$ and the $j^{\text{th}}$ level node $n_j^e$ satisfies the local constraints ($\min_{j-1} \le i-e-1 \le \max_{j-1}$), node $n_j^i$ will be created in the $j$+1$^{th}$ level and the node will be added in the tail of the $j$+1$^{th}$ level and the number of the $j$+1$^{th}$ level nodes should be increased by 1.

**Rule 3. Creation of a parent-child relation between nodes $n_{j-1}^q$ and $n_j^i$.**

If the distance between nodes $n_j^i$ and $n_{j-1}^q$ satisfies the local constraints ($\min_{j-1} \le i-q-1 \le \max_{j-1}$), a parent-child relation between nodes $n_j^i$ and $n_{j-1}^q$ will be created.

The algorithm of NAMEIC is given as follows. Lines 5 through 8 create a root. Lines 9 through 22 create a higher level node. Lines 15 through 19 create a parent-child relationship. Line 20 computes the number of occurrences $N(S,P)$. According to the algorithm, while the Nettree for the problem is created, $N(S,P)$ can be computed.

Algorithm NAMEIC

Input: $S= s_0 s_1 \cdots s_i \cdots s_{n-1}$ and $P=p_0[\min_0,\max_0]\, p_1 \cdots p_{j-1}[\min_{j-1}, \max_{j-1}]\, p_j \cdots p_{m-2}[\min_{m-2}, \max_{m-2}]\, p_{m-1}$

Output: The number of occurrences

Method:

```
1:create the array of hd according to P and compute W
2:sum=0;
3:for (i=0;i<n;i++)
4:    for (j=0;j<m;j++)
5:      if (s[i] == p[j] && j==0) then              //satisfies Rule 1
6:          create a new node nda( n_1^i ) with path 1;
7:          add nda to the tail of the first level of Nettree;
8:      end if
9:      if (s[i] == p[j] && j!=0) then
            // other levels except the first level
10:         ndb= find the first node which satisfies local constraints
[min_{j-1}, max_{j-1}] according to hd[j-1].start;
11:         hd[j-1].start=ndb;                      //update hd[j-1].start
12:         if (min_{j-1} <=i-ndb.position-1) then // satisfies Rule 2
13:             create a new node nda( n_{j+1}^i );
14:             add nda to the tail of the j+1th level of Nettree;
15:             while (i- ndb.position-1>= gap[j-1].min)
                    // satisfies Rule 3
16:                 create parent-child relationship between ndb and nda
17:                 nda. path += ndb. path ;
18:                 ndb = ndb.next ;
19:             end while
20:             if (j==m-1) then sum+= nda. path;
                //compute the number of occurrences
21:         end if
22:     end if
23:   end for
24:end for
25:return sum;
```

### 3.4. Correctness

In this subsection, we prove the correctness of NAMEIC.

**Theorem 1.** The number of root paths of a certain node is the sum of root paths of its parents.

**Proof** (Proof by induction.)

Assume node $n_2^t$ has $k$ parents, $n_1^{t_1}$, $n_1^{t_2}$, $\cdots$, $n_1^{t_k}$. There are $k$ kinds of different root paths which are $\{n_1^{t_1}, n_2^t\}$, $\{n_1^{t_2}, n_2^t\}, \cdots, \{n_1^{t_k}, n_2^t\}$. Therefore $Path(n_2^t)$ is $k$. Because $Path(n_1^{t_1}) = Path(n_1^{t_2}) = \cdots = Path(n_1^{t_k}) = 1$, equation (2) is correct.

Assume the nodes of the $j^{th}$ level and above levels satisfy equation (2), the $j+1^{th}$ level nodes should also satisfy equation (2).

Node $n_{j+1}^t$ has $k$ parents, $n_j^{t_1}$, $n_j^{t_2}$, $\cdots$, $n_j^{t_k}$. So there are $k$ kinds of different paths from the $j^{th}$ level nodes to node $n_{j+1}^t$ which are $\{n_j^{t_1}, n_{j+1}^t\}$, $\{n_j^{t_2}, n_{j+1}^t\}$, $\cdots$, $\{n_j^{t_k}, n_{j+1}^t\}$. So there are $Path(n_j^{t_1})$, $Path(n_j^{t_2})$, $\cdots$, $Path(n_j^{t_k})$ kinds of different root paths from the first level nodes to node $n_{j+1}^t$ and passing through nodes $n_j^{t_1}$, $n_j^{t_2}, \cdots, n_j^{t_k}$ respectively. Therefore equation (2) is correct.

This completes the proof.

Assuming there are $k$ nodes in the $m^{th}$ level of the Nettree for the problem, these node names are $n_m^{t_1}$, $n_m^{t_2}$, $\cdots$ $n_m^{t_k}$. So there are $Path(n_m^{t_1})$, $Path(n_m^{t_2}), \cdots$, $Path(n_m^{t_k})$ kinds of different root paths from roots to the $m^{th}$ level nodes. Hence, according to Theorem 1, $N(S,P)$ is the sum of root paths of all $m^{th}$ level nodes, and it can be written as

$$N(S,P) = \sum_{i=1}^{k} Path(n_m^{t_i})\,. \qquad (3)$$

### 3.5. An illustration example

In this subsection, an illustration example is used to show how our NAMEIC works.

**Example 1**. Given the sequence $S$= abababaa and pattern $P$= a[0,3]b[0,4]a[0,3]a, we have $min_0$=0, $max_0$=3, $min_1$=0, $max_1$=4, $min_2$=0 and $max_2$=3.

A Nettree can be created and the result is shown in Fig. 3. We can see that each index $i$ is created as many nodes in the Nettree. For example, index 4 is created as three nodes in the Nettree, $n_1^4$, $n_3^4$ and $n_4^4$ because $s_4$=a can match $p_0$=a, $p_2$=a and $p_3$=a. Similarly, we can know that indices 2, 5 and 6 are created as two nodes, three nodes and three nodes respectively.

$Path(n_1^0) = 1$ because node $n_1^0$ is a first level node. $Path(n_2^1) = 1$ because node $n_2^1$ has only 1 parent, node $n_1^0$. $Path(n_2^3) = 2$ because node $n_2^3$ has 2 parents, nodes $n_1^0$ and $n_1^2$, and $Path(n_1^0) = Path(n_1^2) = 1$. Similarly, we can know that root paths of nodes $n_4^4$, $n_4^5$ and $n_4^6$ are 1, 4 and 7 respectively. $N(S,P)$ is 12 because nodes $n_4^4$, $n_4^5$ and $n_4^6$ are three fourth level nodes of the Nettree and 1+4+7=12. All occurrences are {0, 1, 2, 4}, {0, 1, 2, 5}, {0, 1, 4, 5}, {0, 3, 4, 5}, {2, 3, 4, 5}, {0, 1, 2, 6}, {0, 1, 4, 6}, {0, 3, 4, 6}, {2, 3, 4, 6}, {0, 1, 5, 6}, {0, 3, 5, 6} and {2, 3, 5, 6}. The problem is solved after creating the Nettree in one pass.

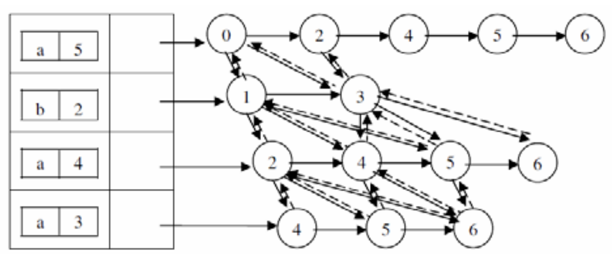


Figure 3. The Nettree for example 1.

## 4. Analysis

### 4.1. Complexity

The number of the Nettree levels is $O(m)$. The space complexity of all Nettree nodes is $O(W*m*n)$ because the depth of the Nettree is $m$, each level has no more than $n$ nodes and each node has no more than $W$ parents and $W$ children, where $m$, $n$ and $W$ are the lengths of pattern $P$ and sequence $S$ and the maximal gap flexibility respectively. Therefore the space complexity of NAMEIC is $O(W*m*n)$.

Meanwhile the space complexity of NAMEIC can be reduced to $O(W*m+n)$. If $i > W+1$, all nodes of $n_j^{i-w-2}$ $(1 \le j \le m)$ of each level can be deleted because these nodes are useless for creating new nodes. Therefore after adding the following code before line 23 of algorithm NAMEIC, NAMEIC-INT (Incomplete NetTree) can be realized.

| |
|---|
| if $i > W+1$ then delete node ( $n_{j+1}^{i-W-2}$ ) |

It is easy to know that the time complexity of both NAMEIC and NAMEIC(INT) are $O(W*m*n)$.

[12] gives the time complexity of PAIG-RST (Reduced Space and Time) which is $O(W^2*m^2*n)$. The comparisons of time and space complexity for PAIG, NAMEIC and NAMEIC(INT) are shown in Table 1.

From Table 1, the time complexity of NAMEIC is $1/(W*m)$ times of PAIG-RST and an example is given below to illustrate the difference between PAIG-RST and NAMEIC.

Table 1. The comparisons of time and space complexity for each algorithm

| Algorithm | Time complexity | Space complexity |
|---|---|---|
| PAIG-RST [12] | $O(W^2*m^2*n)$ | $O(W*m+n)$ |
| NAMEIC | $O(W*m*n)$ | $O(W*m*n)$ |
| NAMEIC(INT) | $O(W*m*n)$ | $O(W*m+n)$ |

**Example 2.** Let us consider the same problem in example 1. Here the problem is solved by PAIG-RST.

Tables 2 and 3 are constructed by PAIG-RST to solve the problem. $N(P,S)$ =1+3+5+1+2=12 according to the cells "4(1),5(3),6(5)" and "5(1),6(2)" in Table 3. According to these two tables, some position indices are computed many times. For example indices 5 and 6 are computed twice in the last column of Table 3.

Table 2. Matching lookup table

| Index | S | a[0,3]b | b[0,4]a | a[0,3]a |
|---|---|---|---|---|
| 0 | a | 1,3 | - | 2,4 |
| 1 | b | - | 2,4,5 | - |
| 2 | a | 3 | - | 4,5,6 |
| 3 | b | - | 4,5,6 | - |
| 4 | a | - | - | 5,6 |
| 5 | a | - | - | 6 |
| 6 | a | - | - | - |

Table 3. Matching table for prefix patterns

| Index | S | a[0,3]b | a[0,3]b[0,4]a | a[0,3]b[0,4]a[0,3]a |
|---|---|---|---|---|
| 0 | a | 1(1) | 2(1),4(2),5(2),6(1) | 4(1),5(3),6(5) |
| 1 | b | - | - | - |
| 2 | a | 3(1) | 4(1),5(1),6(1) | 5(1),6(2) |
| 3 | b | - | - | - |
| 4 | a | - | - | - |
| 5 | a | - | - | - |
| 6 | a | - | - | - |

### 4.2. Analysis of the problem

Let the maximal length of all occurrences be a constant. It can be expressed by the following equation.

$$1+max_0+1+max_1+1+\cdots + max_{m-2}+1=C_1. \quad (4)$$

We know that

$$N(S,P)<n*(max_0-min_0+1)* \cdots * (max_{m-2}-min_{m-2}+1). \quad (5)$$

To maximize the upper bound of $N(S, P)$, we must set

$$max_0=max_1=\cdots =max_{m-2}, \quad (6)$$

$$min_0=min_1=\cdots=min_{m-2}=0. \quad (7)$$

Let $max_0=max_1=\cdots =max_{m-2}=T$, equations (4) and (5) can be written as equations (8) and (9) respectively.

$$(m-1)*(T+1)=C_2, \quad (8)$$

where $C_2=C_1-1$.

$$N(S,P)<n*(T+1)^{(m-1)}. \quad (9)$$

$n$ is a given value and it can be neglected. To maximize $(T+1)^{(m-1)}$ means to maximize $(m-1)\log(T+1)$. Assuming $W=T+1$ , equation (8) can be written as

$$m-1=C_2/W. \quad (10)$$

Let $F(W)= \log(W)\ C_2/W$. The derivate of $F(W)$ is equation (11).

$$F'(W)= C_2/W^2\ (1/W*W-1*\log(W)). \quad (11)$$

If $F'(W)=0$, $\log(W)=1$ i.e., $W \approx 2.718$. As $W$ should be an integer, we have $W=3$ and $T=2$. It means that when $\forall j$: $min_{j-1}=0$ and $max_{j-1}=2$ ($0< j \leq m-1$), $N(S,P)$ can get the maximum value and it is an exponential problem for $m$.

## 5. Experiments

In this section, three kinds of experiments are carried out. All experiments are conducted on a computer with Pentium (R) 4 CPU 3.40GHZ and 1.0 GB of RAM, Windows XP. In these experiments, all sequences $S$ are "aa⋯a⋯a", but the length of $S$ may be different and pattern $P$=a[0,$T$]a[0,$T$]a can be denoted as "a $([0,T]a)^2$".

The first experiment verifies that when the maximal length of all occurrences is a constant and $\forall$ j: $min_{j-1}=0$ and $max_{j-1}=2$ ($0< j\leq m-1$), $N(S,P)$ can get the maximum value. The maximal distances for all patterns are 25 and the results are given in Table 4. We can see that when the pattern is "a$([0,2]\ a)^8$", $N(S,P)$ gets the maximum value 734832. Therefore it verifies when $min_{j-1}=0$ and $max_{j-1}=2$, $N(S,P)$ can get the maximum value.

Table 4. The upper bound of $N(S,P)$ testing

| The length of S | The pattern | The maximal length of all occurrences | $N(S,P)$ |
|---|---|---|---|
| 128 | a$([0,1]a)^{12}$ | 25 | 450560 |
| 128 | a$([0,2]a)^{8}$ | 25 | 734832 |
| 128 | a$([0,3]a)^{6}$ | 25 | 462848 |
| 128 | a$([0,5]a)^{4}$ | 25 | 147744 |
| 128 | a$([0,7]a)^{3}$ | 25 | 58624 |
| 128 | a$([0,11]a)^{2}$ | 25 | 16560 |
| 128 | a$([0,23]a)^{1}$ | 25 | 2772 |

In order to verify that $N(S,P)$ is an exponential problem for $m$, a group of experiments and the results are given in Table 5, where the rate is $N(S,a([0,2]a)^{m+1})/ N(S, a([0, 2]a)^{m})$ . We can see that all rates are closed to 3 because $max_{j-1}-min_{j-1}+1$ is 3. The results show that the problem is an exponential problem for $m$. All entries in Tables 4 and 5 are solved in 0 ms and so the time cost column is neglected.

Table 5. The exponent problem for $m$ testing

| The length of S | The pattern | $N(S,P)$ | Rate |
|---|---|---|---|
| 128 | a$([0,2]a)^{2}$ | 1116 | - |
| 128 | a$([0,2]a)^{3}$ | 3294 | 2.9516 |
| 128 | a$([0,2]a)^{4}$ | 9720 | 2.9508 |
| 128 | a$([0,2]a)^{5}$ | 28674 | 2.9500 |
| 128 | a$([0,2]a)^{6}$ | 84564 | 2.9492 |
| 128 | a$([0,2]a)^{7}$ | 249318 | 2.9483 |

In order to compare the time cost between NAMEIC and PAIG-RST, *N(S,P)* is neglected. Since it is impossible to give a precise time cost less than 16 ms, " $\leq 16$ " is used to indicate less than 16 ms. Looking at the results in Table 6, NAMEIC is clearly faster than PAIG-RST.

Table 6. A comparison of time cost of NAMEIC and PAIG-RST

| The length of *S* | The pattern | Time cost of NAMEIC(ms) | Time cost of PAIG-RST(ms) |
|---|---|---|---|
| 256 | $a([0,2]a)^{10}$ | $\leq 16$ | $\leq 16$ |
| 256 | $a([0,2]a)^{11}$ | $\leq 16$ | $\leq 16$ |
| 256 | $a([0,2]a)^{20}$ | $\leq 16$ | 31 |
| 256 | $a([0,2]a)^{40}$ | 16 | 94 |
| 256 | $a([0,2]a)^{80}$ | 47 | 453 |
| 512 | $a([0,2]a)^{80}$ | 78 | 1328 |

## 6. Conclusions

Pattern matching with flexible wildcard constraints is a more difficult problem than the classical one because the length of wildcards is a range in the new problem whereas it is a constant in the traditional problem. In this paper, a new algorithm (NAMEIC) based on Nettree has been proposed to solve pattern matching with flexible wildcard constraints more effectively and the time complexity of algorithm is $O(W*m*n)$, where $n$ is the length of $S$ , $m$ is the length of $P$ and $W$ is the maximal gap flexibility. NAMEIC can give its result after creating a Nettree in one pass.

The main contribution of this paper is to propose a new nonlinear structure, Nettree. A Nettree is different from a tree in that a node may have more than one parent. Since it is a new kind of data structure, it may have many special concepts and properties. We believe that not only the method can be applied in many fields such as stream data mining, biological sequence analysis, text indexing, time series data mining and so on but also Nettree will attract more theoretical studies.

## References


[1] Cole, J. R., Chai, B., Farris, R. J., Wang, Q., Kulam, S. A., McGarrell, D. M., Garrity, G. M., and Tiedje, J. M., "The Ribosomal Database Project (RDP-II): sequences and tools for high-throughput rRNA analysis", Nucleic Acids Research, 2005, Vol. 33, pp. 294-296.

[2] He, Y., Wu, X., Zhu, X. and Arslan, AN, "Mining Frequent Patterns with Wildcards from Biological Sequences", Proceedings of the 2007 IEEE International Conference on Information Reuse and Integration (IEEE IRI-07), 2007, pp. 329-334.

[3] Cole, R., Gottlieb, L.A., Lewenstein, M., "Dictionary matching and indexing with errors and don't cares", Proceedings of the 36th ACM Symposium on the Theory of Computing. 2004, pp. 91-100.

[4] Liu, Y., Wu, X., Hu, X., Gao, J., Wu,G., "Pattern matching with wildcards based on key character location". Proceedings of the 2009 IEEE International Conference on Information Reuse and Integration (IEEE IRI-09), 2009, pp.167-170.

[5] Faloutsos, C., Ranganathan, M., and Manolopoulos, Y., "Fast subsequence matching in time-series databases", Proceedings of ACM-SIGMOD International Conference on Management of Data (SIGMOD). 1994,Vol23 , Issue 2, pp. 419 - 429.

[6] Fan, W., Wang, H., Yu, P. S., "System and method for tree structure indexing that provides at least one constraint sequence to preserve query-equivalence between xml document structure match and subsequence match", United States Patent 7475070. 2009.

[7] Moon, Y. S., "Efficient stream sequence matching algorithms for handheld devices on time-series stream data", Proceedings of the 24th IASTED international conference on Database and applications table of contents. 2006, pp. 44-49.

[8] Eisinger, J., Klaedtke, F., "Don't Care Words with an Application to the Automata-based Approach for Real Addition", Formal Methods in System Design. 2008, pp. 85-115.

[9] Linhart, C., Shamir, R., "Matching with don't-cares and a small number of mismatches", Information Processing Letters. 2009,109(5). pp. 273-277.

[10] Fischer, M.J., Paterson, M.S., "String matching and other products", Karp RM(ed) Complexity of computation, 1974, vol 7. pp. 113-125.

[11] Chen, G., Wu, X., Zhu, X., Arslan, A. N., and He, Y., "Efficient string matching with wildcards and length constraints". Knowledge and Information Systems, 2006, vol. 10, no. 4, pp. 399-419.

[12] Min, F., Wu, X., Lu, Z., "Pattern matching with independent wildcard gaps". Proceedings of the 8th International Conference on Pervasive Intelligence and Computing . 2009, pp.194-199.

[13] Huang, Y., Wu, X., Hu, X., Xie, F., Gao, J., Wu, G., "Mining Frequent Patterns with Gaps and One-off Condition", Proceedings of the 12th IEEE International Conferences on Computational Science and Engineering. 2009, pp. 180-186.

[14] Zhu, X., Wu, X., "Mining Complex Patterns across Sequences with Gap Requirements", Proceedings of the 20th International Joint Conference on Artificial Intelligence. 2007, pp. 2934-2940.