Solving Flipping Coin Problem and Penney-Ante Game without Using Generating Function

Kai Jin

Institute for Theoretical Computer Science, Tsinghua University
Beijing, 100084, P. R. China, cscjjk@gmail.com

Abstract. We revisit the Flipping Coins (FCP) Problem and Penny-Ante Game(PAG) and present a new method for solving them. In FCP and PAG, we consider the process where a string S is generated character by character randomly until certain fixed pattern occurs. FCP asks the expected length of the S while PAG asks the probability of occurrence for each pattern.
The classical way (see [13]) of computing those statistic quantities was based on generating function(g-f) with elegant formulas for solving FCP and PAG. Suppose there are t patterns and their total length is L. It requires $O(t^3+t\cdot L)$ running time, in which $O(t^3)$ is for solving t variables linear equations, and $O(t\cdot L)$ is for computing the coefficients.
We give a more intuitive way to solve FCP and PAG. Our algorithm is based on the Markov-chain model, without using any g-f techniques. It combines several basic algorithms such as KMP and DFS, and its running time is also $O(t^3+t\cdot L)$. The accuracy of the answer is better then the previous method.

Key words: Flipping Coins, Penney-Ante Game, Pattern Occurrence, Waiting Time, Linear Equations, (Partial match Function) Failure Function, Trie Tree, Markov Chain

1 Introduction

1.1 Problem Description of FCP and PAG

Suppose we generate a string $S = x_1x_2x_3...$ one character at a time, where each x_i is chosen independently from $\Sigma = \{1, 2, .., n\}$. $\forall 1 \le i \le n$, the probability for choosing i is $p_i, 0 < p_i < 1, \sum_i p_i = 1$. The generation stop as soon as any fixed pattern M_i among $\{M_1, M_2, .., M_t\}$ occurred as a substring(actually a suffix) of S. We say the occurred pattern *wins*.

$M_i(1 \le i \le t)$ is a non-empty string consisting of characters belongs to Σ. The lengths of patterns could be distinct, but to avoid that more than one pattern wins, no pattern can be a suffix of another one (for simplification, unnecessary). We also assume no pattern is a prefix of another one, otherwise some pattern would never win.

Now given $n, p_1, p_2, .., p_n$ and $M_1, M_2, .., M_t$, the Flipping Coin Problem (FCP) asks for the expected length of S; Penny-Ante Game (PAG) asks for the winning probability win_i of pattern M_i.

We can find many applications of FCP and PAG in the biological information area. See [15], [9] and [10].

1.2 Alphabet and Probabilistic Distribution

Originally, FCP and PAG consider consecutive random results of tossing a fair coin or a fair penny, so the alphabet size is 2 and the probability is $\frac{1}{2}$ for each outcome. But it turns out that in the g-f based algorithms the alphabet doesn't matter, and the size could be very large and the distribution could be non-uniform, see [13].

1.3 Related work

PAG was a real game first invented by Walter Penney [1] in 1969 and mainly focused on two patterns at that time.

FCP and PAG have been extensively studied. Many result focused on investigating the way of choosing patterns so that the winning probability is optimal, see [2],[6],[7],[8]. For 2-players Penney-Ante Game, a remarkable result says that a player could always win the game at about 2/3 if the other player's pattern is known, see [7] and [5]. (Both should pick up string with the length $len \geq 3$) Then it is definitely a non-transitive game, see [6]. Also, there are lots of other statistic quantities beside expected length and winning probabilities, so many result focused on finding generating function, properties and also algorithms for them, see [10], [11], [12].

In this article, we focused on the basic computing problem. An easy idea is to construct a Markov-chain and solve a L variables linear equation which need $O(L^3)$ time, see [14]. But it's hard to compute the answer fast based on this idea. John H. Conway [6] first gave a beautiful formula for the 2-players PAG. It can be proved in various way, see [4] and [2]. The method can be generated to multiply players, see [13]. The technique is mainly based on generating function. It need to solve a linear equations which requires $O(t \cdot L)$ to compute the coefficients and $O(t^3)$ time to solve the equations.

1.4 Our Results

As in previous algorithms (see [13]), our algorithm also need to solve t variables linear equations. But our equations are different from the previous one. We use a more instinctive way for obtain the equations. We only uses several basic algorithm such as KMP and DFS, and we solve the equations in the order that the precision of the answer seems better than the original one since we choose the main element in the elimination more rational.

It will be shown that the time complexity is $O(t^3 + t \cdot s)$, here s is the number of nodes in the Markov-chain. In best case, $s \ll L$; in worst case, $s = \Omega(L)$.

1.5 Outline

The multi-pattern case is much more complicated than the one pattern case. In section 2, we analyze the one pattern case, and give a detailed algorithm. In section 3, we extend this algorithm to multi-pattern case. In the one pattern case, PAG is trivial so we only consider FCP. In the multi-pattern case, we consider PAG problem. FCP can be solved by modifying the constants in the equation since no big difference exists between FCP and PAG. Finally, we summarize the idea of our algorithm, and give some open questions related.

2 One Pattern Case

2.1 Model: NFA–Simple Chain

Denote $m = |M|$, assume $M = M_1M_2..M_m$, denote $M^{(i)}$ to be $M[1..i]$ which is the i-th prefix of M. The generating process of string S could be described as a $m + 1$ states NFA(Non-deterministic Finite Automata) as shown in Fig. 1.

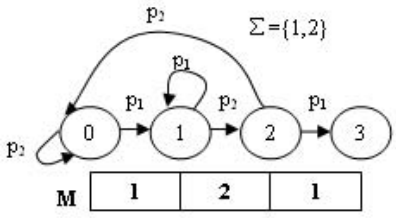

Fig. 1. The generating process of string S could be described as a $m + 1$ states NFA.

All states are represented by nodes labeled from 0 to m, state m is the terminal state. For each state $i(0 \leq i \leq m-1)$, there are n outgoing edges whose ending points are denoted by $e_{i,1}, e_{i,2}, .., e_{i,n}$. Here $e_{i,k}$ means the next state the NFA should go if it's at state i, and the next character is k.

$$e_{i,k} = max\{j | M^{(j)} \texttt{is a suffix of} M^{(i)} +' k'\}. \quad (1)$$

Each edge has an associated probability. The k-th outgoing edge of each node is equal to p_k.

This model is an instance of Markov-chain. Our goal is to calculate the stopping time of this Markov-chain.

2.2 Linear Equations

Denote x_i as the expected stopping time starting from node i. We know $x_m = 0$, and our goal is to get x_0. The nature way for getting it is solving the following

m equations in which $Q_{i,j}$ represent the sum of the associated probabilities of all edges from node i to node j.

$$x_i = 1 + \sum_{j=0}^{i+1} Q_{i,j} \cdot x_j \, (0 \le i < m) \tag{2}$$

We know that these equations have exactly one solution. But we can't solve it efficiently because the coefficients matrix is not triangular. The error could be huge if we calculate it directly.

In the next subsection, we first make a simple linear combination from $x_0, .., x_{m-1}$ to $u_0, .., u_{m-1}$, then point out that it's easy to either calculate u, or calculate x according to u.

2.3 Triangular Transform Technique

$\forall 0 \le i < m$,

$Q_{i,i+1} + \sum_{j=0}^{i} Q_{i,j} = 1$
$\Rightarrow \left(Q_{i,i+1} + \sum_{j=0}^{i} Q_{i,j}\right) x_i = 1 + Q_{i,i+1} x_{i+1} + \sum_{j=0}^{i} Q_{i,j} x_j$
$\Rightarrow Q_{i,i+1} \left(x_i - x_{i+1}\right) = 1 + \sum_{j=0}^{i} Q_{i,j} \left(x_j - x_i\right)$
let $u_i = x_i - x_{i+1} (0 \le i < m)$, then those m equations can be transformed into:

$$Q_{i,i+1} u_i = 1 + \sum_{j=0}^{i} Q_{i,j} \sum_{k=j}^{i-1} u_k \tag{3}$$

Here $Q_{i,i+1} > 0$, there is a proof in the next subsection, see Lemma 1 (6).

The coefficient matrix of u is triangular. If Q can be computed efficiently, we can compute $u_i (0 \le i < m)$ in $O(m \cdot n)$ time. Then, from $x_0 - x_m = u_0 + u_1 + .. + u_{m-1}$ and $x_m = 0$ we can compute x_0. Actually we can compute $x_i (0 \le i < m)$ from $u_i (0 \le i < m)$ in $O(m)$ time.

In the next few subsections, we will focus on how to compute or represent $Q_{i,j}$, and then we will find out that (3) actually can be computed much faster.

2.4 Basic Properties of Q

Let $d_{i,j}$ be the number of edges from node i to node j. Since the probability of any edge ending at node $j (j > 0)$ must be associated with probability p_{M_j}, so

$$Q_{i,j} = \begin{cases} d_{i,j} \cdot p_{M_j} & j > 0 \\ 1 - \sum_{j \ge 1} d_{i,j} \cdot p_{M_j} & j = 0 \end{cases} \tag{4}$$

$d_{i,j}$ can be represented by $e_{i,j}$, that is, $d_{i,j} = |\{k | 1 \le k \le n, e_{i,k} = j\}|$.

Lemma 1 (Constraints of $d_{i,j}$).

$$\forall j > 0, 0 \le d_{i,j} \le 1 \ ; \tag{5}$$
$$\forall 0 \le i < m, d_{i,i+1} = 1 \ ; \tag{6}$$
$$\forall j > i+1, d_{i,j} = 0 \ . \tag{7}$$

Proof. According to (18)

(5): Suppose $d_{i,j} \ge 2$, $\exists k_1 \ne k_2, s.t. e_{i,k_1} = e_{i,k_2} = j > 0$, then $M^{(j)}$ is a suffix of $M^{(i)} +' k_1'$ and $M^{(i)} +' k_2'$ simultaneously. $k_1 = k_2$, contradiction!

(6): $M^{(i+1)}$ is the longest suffix of $M^{(i)} + M_{i+1}$
$e_{i,M_{i+1}} = i+1 \therefore d_{i,i+1} = 1$.

(7): $\because 0 \le e_{i,k} \le i+1 \therefore \forall j > i+1, d_{i,j} = 0$. □

Now applying Lemma 1 to (3), we get

$$\begin{aligned} p_{M_{i+1}} u_i &= 1 + \textstyle\sum_{j \in D_i} p_{M_j} \sum_{k=j}^{i-1} u_k + \left(1 - p_{M_{i+1}} - \sum_{j \in D_i} p_{M_j}\right) \sum_{k=0}^{i-1} u_k \\ &= 1 + \textstyle\sum_{j \in D_i} p_{M_j} (y_i - y_j) + \left(1 - p_{M_{i+1}} - \sum_{j \in D_i} p_{M_j}\right) y_i \\ &= - \left[\textstyle\sum_{j \in D_i} p_{M_j} y_j\right] + \left[1 + \left(1 - p_{M_{i+1}}\right) y_i\right] \end{aligned} \tag{8}$$

Here $D_i = \{j | 1 \le j \le i, d_{i,j} = 1\}$ and $y_i = \sum_{j=0}^{i-1} u_j = x_0 - x_i$

If we compute u_i in the order from u_0 to u_{m-1}, y_i could be computed in $O(1)$ time because $y_i = y_{i-1} + u_{i-1}$, hence the difficulty is how to compute the first part $f_i = \sum_{j \in D_i} p_{M_j} y_j$ efficiently.

Now we want a specific representation of D_i.

2.5 Connections between KMP Prefix Function π_i and set D_i

Definition 1 (prefix function). *$\pi : \{0, .., m\} \to \{0, .., m\}$ is the prefix function of M, if $M^{(\pi_i)}$ is the longest suffix of $M^{(i)}$ except itself among all prefixes of M. concretely,*

$$\pi_i = \begin{cases} 0 & i = 0 \\ max\left\{j < i | M^{(j)} \texttt{is a suffix of} M^{(i)}\right\} & i > 0 \end{cases} \tag{9}$$

We also extend this definition to

$$\pi_i^{(k)} = \begin{cases} i & k = 0 \\ \pi_i & k = 1 \\ \pi_{\pi_i^{(k-1)}} & k > 1 \end{cases} \text{ and } \Pi_i = \left\{\pi_i^{(h)} | h \ge 0\right\}$$

The prefix function π can be computed in $O(n+m)$ time by using the famous KMP algorithm.

Lemma 2 (Π_i and Π_{π_i} is similar).

$$\forall i > 0, \Pi_{\pi_i} = \Pi_i - \{i\} \tag{10}$$

Lemma 3 (Suffix of $M^{(i)}$, Prefix of M).

$$M^{(j)} \texttt{is a suffix of} M^{(i)} \iff j \in \Pi_i \tag{11}$$

Lemma 4 (Using Π_i to Describe D_i).

$$e_{i,k} = \begin{cases} \max(H_i') & H_i' \neq \emptyset \\ 0 & otherwise \end{cases}, H_i' = \{h | h-1 \in \Pi_i, M_h = k\} \tag{12}$$

$$d_{i,j}(j>0) = \begin{cases} 1 & \max(H_i'') = j \\ 0 & otherwise \end{cases}, H_i'' = \{h | h-1 \in \Pi_i, M_h = M_j\} \tag{13}$$

$$D_i = \{1 \leq j \leq i | j-1 \in \Pi_i, \forall\left(h > j, h-1 \in \Pi_i,\right), M_h \neq M_j\} \tag{14}$$

Proof. (12): From formula (18), $e_{i,k} = \max\{j | M^{(j)} \texttt{is a suffix of} M^{(i)} +' k'\} = \max\{j | M^{(j-1)} \texttt{is a suffix of} M^{(i)}, M_j = k\} = \max\{j | j-1 \in \Pi_i, M_j = k\}$,

(13): It's equivalent to (12).

(14): $j \in D_i \iff d_{i,j} = 1 \iff \max\{h | h-1 \in \Pi_i, M_h = M_j\} = j \iff j-1 \in \Pi_i, \forall\left(h > j, h-1 \in \Pi_i\right), M_h \neq M_j$

□

Lemma 5. *The following method computes* D_i*,*

Start $Seq^{(i)} = \left(\pi_i + 1, \pi_i^{(2)} + 1, .., \pi_i^{(j)} + 1, ..\right)$*, here* $Seq_j^{(i)} = \pi_i^{(j)} + 1 (\forall j \geq 1)$*;*
Remove $\forall (Seq_j^{(i)}, Seq_k^{(i)})$ *s.t.* $j < k$ *and* $M_{Seq_j^{(i)}} = M_{Seq_k^{(i)}}$*, remove* $Seq_k^{(i)}$*;*

Proof. This is just another words to say (14). □

2.6 Constant Difference, Depth First Search

Theorem 1. *The following method can also compute* D_i*,*

Call Call the process for D_{π_i}*, let* $Seq^{(i)} = Seq^{(\pi_i)}$*;*
Insert put $\pi_i + 1$ *in front of the sequence* $Seq^{(i)}$*;*
Remove If $\exists Seq_k^{(i)}$ *s.t.* $k > 1$ *and* $M_{Seq_1^{(i)}} = M_{Seq_k^{(i)}}$*, remove* $Seq_k^{(i)}$*;*

Proof. We can prove it by using the equation $Seq^{(i)} = \left(\pi_i + 1, Seq^{(\pi_i)}\right)$ *and Lemma 5.* □

Theorem 1 says we can compute D_i recursively. If we store $Seq^{(i)}$ for each i, it gives an $O(edges^*)$ time way to compute them, here $edges^* = \sum_{i=1}^{m-1} |D_i|$ which is the number of all edges except those whose ending point are state 0 in the NFA. Lemma 6 says $edges^* = O(m+n)$, so this method is efficient indeed.

Theorem 1 also says D_i and D_{π_i} have at most 2 different elements! Because we will insert only one new element π_i+1, and we may remove at most one existing element. Without lose of generalization, assume $D_i = D_{\pi_i} + \{\alpha_i\} - \{\beta_i\}$, then

$$f_i = f_{\pi_i} + p_{M_{\alpha_i}} y_{\alpha_i} - p_{M_{\beta_i}} y_{\beta_i} \tag{15}$$

Using DFS, we can compute α, β in $O(m+n)$ time. During the search, we need to maintain an $O(n)$ size Reverse-Index array R dynamically, here $R_k = \begin{cases} j & (D_{i,j}=1) \wedge (M_j = k) \\ nil & otherwise \end{cases}$. We don't have to store $Seq^{(i)}$ for each i.

2.7 Algorithm

- Use KMP algorithm to get π;
- Build up the π-tree $(G = \{0, .., m-1\}, E = \{(i,j)|\pi_i = j\})$;
- DFS in the π-tree to get α, β;
- Calculate u in index increasing order u_0 to u_{m-1}, according to (8) and (15).
- Calculate x from u.

Each steps take $O(n+m)$ time, so the algorithm runs in linear time.

In this approach, we don't need the truth described in Lemma 6. But we've already discussed what it means at the analysis after Theorem 1. Now we give a proof of it, this proof is first shown by Zhiyi Huang.

Lemma 6.

$$edges^* = O(m) \tag{16}$$

Proof. First, $|\{(i \to 0), (i \to 1), (i \to i+1)\}| \leq 3m$.

Now suppose there is an edge $(i \to j)$, $2 \leq j \leq i$. *Let* $k = i - j$, *then*

$$M[k+2] = M[1], M[k+3] = M[2], .., M[k+j] = M[j-1], M[k+j+1] \neq M[j] \tag{17}$$

$\therefore \forall k$ *that is fixed, there is at most one* j *fulfil (17), hence there is at most one edge* (i,j) *such that* $i - j = k$. □

3 Multi-pattern Case

Situation becomes much more difficult in multi-pattern case. Patterns would interfere each other. All patterns constitute into an integrated system which can not be isolated to several single pattern. So it's nontrivial to find out how to extend our algorithm in section 2 to the multi-patterns case. We will give an extension in this section, but the time complexity is no longer linear.

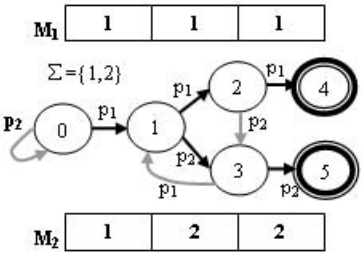

Fig. 2. The generating process of string S could be described as trie-like NFA.

3.1 Model: NFA–Trie (Prefix Tree)

Denote $m_i = |M_i|$, assume $M_i = M_{i,1}, M_{i,2}, .., M_{i,m_i}$. Since there may be strings with common prefix, the total number of states in the NFA, denoted by s, could be less then $\sum_i m_i$. For some extreme case, $s \ll \sum_i m_i$.

The shortest path from 0 to any other states are unique. All of them buildup a tree T(All the black edges shown in Fig. 2). Define $depth_i$ be the length of the shortest path from 0 to i, and $father_i$ be the precursor of i on the path. All states are labeled from 0 to $s-1$, such that $i \leq j \leftrightarrow depth_i \leq depth_j$.

For each node $i(0 \leq i \leq m-1)$, there are n outgoing edges whose ending points are denoted by $e_{i,1}, e_{i,2}, .., e_{i,n}$. Here $e_{i,k}$ means the next state the NFA should go if it's at state i, and the next character is k. $\forall i > 0$, $\exists c_i$ such that the c_i-th outgoing edge of $father_i$ is $(father_i \to i)$. Those edges whose ending point are $i(i > 0)$ must be with associate with probability p_{c_i}.

The concatenation of all characters along the shortest path from 0 to i, is denoted by $S^{(i)}$, which is a prefix of $M_1, M_2, .., M_t$. So this structure is called a prefix tree, or trie.

$$e_{i,k} = max\{j | S^{(j)} \texttt{is a suffix of} S^{(i)} +' k'\}. \tag{18}$$

Since this formula is the same as the one pattern case, lots of properties of the simple case remain. For example, let $d_{i,j}(j > 0)$ be the number of edges from i to j, $d_{i,j} \in \{0, 1\}$.

3.2 Linear Equations

By setting the different initial value, $win_1, win_2, .., win_t$ are all solutions of the following equations respectively. Denote $w_i(0 \leq i < s)$ as the winning probability starting from state i. $w^* = (w_{z_1}, w_{z_2}, .., w_{z_t})$ is the initial value, here z_i is the label of the terminal state of M_i. $win_i(1 \leq i \leq t)$ is the solution when $w^* = e_i = (\underbrace{0, .., 0}_{i-1}, 1, \underbrace{0, .., 0}_{t-i})$.

$$w_i = \sum_{j=0}^{s-1} Q_{i,j} \cdot w_j \ (0 \leq i < m) \tag{19}$$

Here, $Q_{i,j}$ represent the sum of the associated probabilities of all edges from node i to node j.

3.3 Partial Triangular Transform Technique

As in one pattern case, it's hard to compute (19) directly. We have to apply our technique described in subsection 2.3 to equations (19). Before that, we give some more useful definitions:

Define A_i be the set of all ancestors of i; If $father_j = i$, we say j is a child of i; Define tc_i be the number of children of i, $tc_i = |\{j|father_j = i\}| > 1$; If a state i is non-terminal, we say it is an internal state, and $tc_i > 0$, suppose $child_i$ is the child of i with smallest label; If $tc_i > 1$, we say i is a branch. Branch is the most important concept since it's hard to handle.

Now, for all internal state i,

$$\begin{aligned} \because \{j|Q_{i,j} > 0\} &\subseteq \{j|father_j = i\} \cup \{j|depth_j \leq depth_i\} \\ &\subseteq \{j|father_j = i\} \cup \{j|j < child_i\} \end{aligned}$$

and $\{j|j < child_i\} \cap \{j|father_j = i\} = \phi$,

$$\therefore \sum_{father_j=i} Q_{i,j}\,(w_i - w_j) = \sum_{j<child_i} Q_{i,j}\,(w_j - w_i) \tag{20}$$

As before, we define $u_i = w_{father_i} - w_i$. $y_i = w_0 - w_i = \sum_{j\in A_i} u_j$, so

$$\sum_{father_j=i} Q_{i,j}u_j = \sum_{j<child_i} Q_{i,j}\,(y_i - y_j) \tag{21}$$

Since T is no longer a chain but a tree, there must be branches . So unfortunately, the coefficients matrix of (21) is no longer a triangular matrix. We can't solve (21) unless we bring some variables called *Temporary Unknown*, or T-U for short. Actually, For each branch i, we will use $tc_i - 1$ such T-Us. Since $\Sigma_{tc_i>1}(tc_i - 1) = t - 1$, we'll use $t - 1$ T-Us. T-Us are just another angle of view of some u_i which we can't solve right now and so treated as unknowns temporarily.

Roughly speaking, whole solving mechanism consists of there steps:

Step 1 Compute partial solution of each variable u_i, here partial solution means that we could use the T-Us in it;

Step 2 Use the terminal values $w_{z_1}, w_{z_2}, .., w_{z_n}$ to compute the real value of those T-Uns;

Step 3 Compute final solution of u_i by replace the T-Us by its real value. Use u_i to compute w_0.

Notice that $t \ll s$, in this way the time complexity should be much better than the brute-force way. This mechanism is not needed in one pattern case since there is no branches there. We haven't found out any trick to handle branches, so undetermined coefficients method is borrowed here.

Concretely, for any branch i, we let $TU_i = \{u_j | j \neq child_i, father_j = i\}$. Rewrite (21) as follows:

$$Q_{i,child_i} u_{child_i} = \sum_{j<child_i} Q_{i,j} (y_i - y_j) - \sum_{j \neq child_i, fahter_j = i} Q_{i,j} u_j \qquad (22)$$

Now Step 1 is clear, we can use (22) to solve the partial solution of each u_i, here $u_j \in \bigcup_i TU_i$ are the TUs; For Step 2, there is $t-1$ TUs, and it's easy to find $t-1$ linear independent equations. Here is a way,

$$y_{z_i} - y_{z_1} = w_{z_1} - w_{z_i} (i = 2, 3, .., t)$$

. The left side can be represented by TUs since $u_i = w_{father_i} - w_i$, $y_i = w_0 - w_i$, and the right side is a constant.

By solving this equation once, we could compute a certain $win_i, 1 \leq i \leq t$.

3.4 Example

For the instance shown in Fig. 2, suppose $p_1 = p_2 = \frac{1}{2}$. u_3 is the only TU.

Step 1 $w_0 = \frac{1}{2} w_1 + \frac{1}{2} w_0 \rightarrow u_1 = 0$
$w_1 = \frac{1}{2} w_2 + \frac{1}{2} w_3 \rightarrow u_2 = -u_3$
$u_3 = u_3$
$w_2 = \frac{1}{2} w_4 + \frac{1}{2} w_3 \rightarrow u_4 = u_2 - u_3 = -2u_3$
$w_3 = \frac{1}{2} w_5 + \frac{1}{2} w_1 \rightarrow u_5 = u_3$

For $w^* = (1, 0)$

Step 2: $w_4 = 1, w_5 = 0$. $y_5 - y_4 = w_4 - w_5 = 1 \rightarrow (u_5 + u_3) - (u_4 + u_2) = 1 \rightarrow u_3 = \frac{1}{5}$

Step 3: $u_1 = 0, u_2 = -\frac{1}{5}, u_3 = \frac{1}{5}, u_4 = -\frac{2}{5}, u5 = \frac{1}{5}$.
$w_0 - w_4 = u_1 + u_2 + u_4 = -\frac{3}{5} \rightarrow w_0 = 1 - \frac{3}{5} = \frac{2}{5}$.

For $w^* = (0, 1)$

Step 2': $w_4 = 1, w_5 = 0$. $y_5 - y_4 = w_4 - w_5 = -1 \rightarrow (u_5 + u_3) - (u_4 + u_2) = -1 \rightarrow u_3 = -\frac{1}{5}$

Step 3': $u_1 = 0, u_2 = \frac{1}{5}, u_3 = -\frac{1}{5}, u_4 = \frac{2}{5}, u5 = -\frac{1}{5}$.
$w_0 - w_4 = u_1 + u_2 + u_4 = \frac{3}{5} \rightarrow w_0 = 0 + \frac{3}{5} = \frac{3}{5}$.

Hence $win_1 = \frac{2}{5}, win_2 = \frac{3}{5}$.

3.5 Optimization and Complexity Analysis

Step 1 Use the the same methods used in subsection 2.4, 2.5, 2.6, we can solve partial solution in time $O(s \cdot t) = O(L \cdot t)$;

Step 2, 3 Assume all the TUs are $u_{i_1}, u_{i_2}, .., u_{i_{t-1}}$, let $U = \left(u^{(2)}, u^{(3)}, .., u^{(t)}\right)$, here $u^{(i)}$ is the TUs vector when $w^* = e_i$.
Let A be the coefficient matrix, $A \cdot U = I_{t-1}$, hence U can be computed in time $O(t^3)$. To solve this $t-1$ variable linear equation, there is a rational order to find the main element in the elimination, because we can find an bijection from all equation and and TUs from the trie structure.

4 Comparisons

Both our algorithm and the g-f based algorithm can solve multi-pattern FCP and PAG, both need to solve an t variables linear equation, but coefficients of them differ. Our method compute the coefficients on the Markov-chain model, hence seems more direct and intelligible.

Both of the algorithms need $O(t \cdot L)$ time for computing coefficients and $O(t^3)$ time for solving the equation.

Our algorithm seems more complicated than the g-f based algorithm, but it's more intuitionistic. Besides, it's likely to get a more precise answer since we know more inner structure of the equation.

5 Conclusion

Although it's easy to find a slow $O(L^3)$ algorithm by using the Markov-chain model directly, it's hard to make it fast. By analyzing the structure of Q, we find out relations between D_i and D_{π_i}, then by using KMP prefix function and using DFS we show that the direct method could also run very fast.

We don't need any advanced techniques such as g-f and suffix tree. So people who knows KMP and DFS could easily understand our algorithm.

5.1 Open Questions

1. How to prove in detail why the answer of our method would be more precise?
2. What's the intrinsic connection between our algorithm and the classical one?
3. What's the lower bound of the time complexity of these problems?
4. Is there more applications which could be optimized by using our techniques?

Acknowledgments. This work was supported in part by the National Natural Science Foundation of China Grant 60553001, and the National Basic Research Program of China Grant 2007CB807900,2007CB807901.

References

1. Walter Penney: "Problem 95: Penney-Ante", Journal of Recreational Mathematics 7, 1974

2. L. J. Guibas and A. M. Odlyzko, String overlaps, pattern matching, and nontransitive games, Journal of Combinatorial Theory (A) 30, 1981.
3. Periods in strings, L. J. Guibas and A. M. Odlyzko, J. Comb. Theory A, 30 (1981).
4. S.-Y. R. Li, A martingale approach to the study of occurrence of sequence patterns in repeated experiments, Annals of Probability 8, Dec 1980.
5. D. Stark, First occurrence in pairs of long words: a penney-ante conjecture of pevzner, Combinatorics, Probability, and Computing 4, 1995.
6. Martin Gardner, On the paradoxical situations that arise from nontransitive relations, Scientific American 231,4, 1974
7. Felix, Daniel, Optimal Penney Ante Strategy via Correlation Polynomial Identities, Electron. J. Comb. 13, No. 1, Research paper R35, 2006.
8. J. A. Csirik, Optimal strategy for the first player in the penney ante game, Combinatorics, Probability, and Computing 1, 1992.
9. S. Breen, M.S. Waterman, and N. Zhang, Renewal Theory for Several Patterns. J. Appl. Prob. 22. 1985
10. Sophie Schbath, An Overview on the Distribution of Word Counts in Markov Chains, Journal of Computational Biology. 7(1-2): 193-201, Feb 2000.
11. Isa Cakir , Ourania Chryssaphinou , Marianne Mansson, On a Conjecture by Eriksson Concerning Overlap in Strings, Combinatorics, Probability and Computing, v.8 n.5, p.429-440, Sep 1999.
12. Vladimir Pozdnyakov, On occurrence of subpattern and method of gambling teams, Annals of the Institute of Statistical Mathematics, vol. 60, issue 1, 2008.
13. Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. Concrete Mathematics: a Foundation for Computer Science. Addison-Wesley Publishing Company, 1989.
14. J. M. Cargal, Discrete Mathematics for Neophytes: Number Theory, Probability, Algorithms, and Other Stuff, Chapter 35.
15. J. D. Watson, T. A. Baker, S. P. Bell. Molecular Biology of the Gene, 1977.