数理解析研究所講究録 1426 巻 2005 年 206-211



# ファクターオラクルの誤受理構造の解析

岩崎久史 (Hisashi Iwasaki)
東京工業大学 理学部 情報科学科
Dept. of Information Science, School of Science,
Tokyo Institute of Technology
email: iwasaki2@is.titech.ac.jp

**概要**

Allauzen ら [ACR99] により提案されたファクターオラクルというデータ構造は, 与えられた文字列 $p$ に対して $p$ のすべての部分文字列を少なくとも受理する性質をもつ決定性有限オートマトンである. このデータ構造はそれまで存在していた suffix tree[McC76] などのデータ構造に比べ, 空間計算量という面で大変改良されている. しかしその一方で, $p$ の部分文字列以外も受理してしまうことがある. 本研究ではこれを誤受理と呼び, 誤受理について解析をした. そして, ファクターオラクルが構成中にはじめて誤受理される文字列を生じるときの必要十分条件を与え, 更に文字列 $p$ に対するファクターオラクル ($Oracle(p)$) が誤受理するかどうか $O(|p|)$ で判定するアルゴリズムを提案した.

## 1 はじめに

文字列に対して特定の文字列が出現するかどうか, また出現するならその出現位置をすばやく特定すること (検索) や, 文字列の中に繰り返し出現するような部分文字列を抽出することは, データ検索のみならず, 遺伝子配列や蛋白質のアミノ酸配列の解析, 特徴付けなど幅広く利用されている技術である. これらに利用されるデータ構造には, suffix tree や suffix array[MM90] があるが, 本論文では 1999 年に Allauzen らによって提案されたファクターオラクル [ACR99] についての解析結果を述べている.

ファクターオラクルは線形時間で構成できる決定性有限オートマトンの一つである. また, suffix tree などと比較して構成アルゴリズムが非常に単純であり, 実装の面でもはるかに容易である. 更に, 構成に必要なメモリ空間が他のデータ構造と比べて少ない. suffix tree もファクターオラクルも空間計算量は $O(|p|)$ であるが, suffix tree では実用するという側面から考えると大きすぎる. この点でファクターオラクルは大幅に改善されている.

ファクターオラクルと他のデータ構造の大きな違いとして, 構成に用いられた文字列 $p$ の全ての部分文字列を少なくとも受理するという性質がある. これは, 他のデータ構造が全ての部分文字列だけ受理するという点において異なる. 本研究では, 部分文字列以外が受理 (誤受理) される場合について解析し, 誤受理するかどうか判定する方法として, ファクターオラクルを構成しながらはじめて誤受理される文字列が出現するときの必要十分条件を与えている. はじめて誤受理される位置

を求めることは, そのファクターオラクルが誤受理かどうか判定することと同じである. またこの方法は, 与えられた文字列 $p$ に対して $O(|p|)$ で判定できる.

最後に本論文の構成は, まず第2章でファクターオラクルについて必要な概念や定義, さらに構成法に関して説明する. 第3章では, 本研究の成果である誤受理判定アルゴリズムについて, 判定条件と構成法を述べる. 最後に第4章で結論と今後の課題について述べる.

## 2 ファクターオラクル

### 2.1 用語

$\Sigma$ をアルファベットとし, 集合の要素 $\sigma \in \Sigma$ を文字として文字列全体の集合を $\Sigma^*$ で表す. 文字列 $p$ の長さを $|p|$ で表す.$\epsilon \in \Sigma$ を空文字列とする ($|\epsilon| = 0$). 文字列 $p$ に対し,$|p| = m$ のものを $p = p_1 p_2 \ldots p_m (p_i \in \Sigma, 1 \leq i \leq m)$ または $p = p[1 \ldots m]$ とする.

$p$ のすべての部分文字列の集合を $Fact(p)$ とし, 文字列 $p$ に対するファクターオラクルを $Oracle(p)$ と表す. $Oracle(p)$ は少なくとも Fact($p$) を受理するオートマトンである. 文字列 $p$ がある文字列 $u, v \in \Sigma^*$ を用いて $p = uwv$ と表せるとき $w$ は $p$ の部分文字列であるという. $p = uv (u, v \in \Sigma^*)$ と表せるとき $u$ を $p$ の prefix(接頭辞), $v$ を $p$ の suffix(接尾辞) という. 文字列 $p$ の長さ $i$ の prefix を $\mathrm{pref}_p(i) (= p[1 \ldots i])$ と表す. $\mathrm{pref}_p(i)$ のすべての suffix を suf($i$) と表す. 文字列 $p$ に対して, $p$ のすべての prefix の集合を Pref($p$), すべての suffix の集合を Suff($p$) と表す.

### 2.2 ファクターオラクルの性質

文字列 $p = p_1 p_2 \ldots p_m$ に対するファクターオラクルは次の4つの性質をもつ.

1. 非環式 (自己ループのない).
2. $p$ の全ての部分文字列を受理.
3. 状態数は $m + 1$ 個. 全て受理状態.
4. 遷移数は $m$ 個以上 $2m - 1$ 個以下.

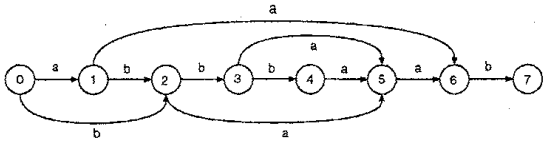


図 1: $Oracle(abbbaab)$

図1は $p = abbbaab$ に対するファクターオラクルである. これを見ると $p$ の全ての部分文字列を確かに受理しているが, 例えば "$aba$" や "$abba$" など部分文字列ではない文字列も受理している.

[ACR99] の中でファクターオラクルの二つの構成法が与えられている. 本研究では online アルゴリズムの方を利用する. まず, 構成に必要な言葉の定義をする.

**定義 1.** $\mathrm{repet}_p(i)$ とは, $\mathrm{pref}_p(i)$ の中で少なくとも2回現れる最長の suffix.

例えば, 図1に示される $p = abbbaab$ の場合, $\mathrm{repet}_p(1) = \epsilon$, $\mathrm{repet}_p(4) = bb$, $\mathrm{repet}_p(7) = ab$ である.

**定義 2.** (Suffix link) $S_p$ は, $Oracle(p)$ の各状態 $i > 0$ から $\mathrm{repet}_p(i)$ が受理される状態 $j$ への写像 ($S_p(i) = j$). $S_p(0) = -1$.

**定義 3.** $Oracle(p)$ の各状態 $i$ に対して,$k_0 = i, k_j = S_p(k_{j-1}) (j \geq 1)$ とする. このとき $SP_p(i) = \{k_0 = i, k_1, \ldots, k_t = 0\}$ とし, これを $Oracle(p)$ の状態 $i$ に対する suffix path と呼ぶ.

$Oracle(p)$ は, $O(p)$ で構成できる. 詳しいアルゴリズムについては [ACR99] にある.

## 3 ファクターオラクルの誤受理判定

この章では, 与えられた文字列 $p$ に対して $Oracle(p)$ が誤受理するかどうかについて考える.

◎ファクターオラクルの誤受理判定問題

入力: 文字列 $p$

質問: $Oracle(p)$ は誤受理するか？

まずはじめに, $Oracle(p)$ が初めて誤受理するときの必要十分条件を与え, 次にこの条件と [LL00] で紹介されている近似的な $|\mathrm{repet}_p(i)|$ を計算するアルゴリズムを用いて判定を行う.

### 3.1 $Oracle(p)$ が初めて誤受理するときの必要十分条件

まずは, 初めて誤受理するということについて説明する. 文字列 $p = p_1p_2 \ldots p_m$ に対して, $Oracle(p_1p_2 \ldots p_i)$ では誤受理しないが $Oracle(p_1p_2 \ldots p_ip_{i+1})$ で誤受理するときを初めての誤受理と呼ぶことにする. $Oracle(p)$ を構成中にある文字を加えることによって誤受理するようになるとその後文字をどんなに加えていっても誤受理が消えることはない. よって, どこかで初めて誤受理することが分かれば $Oracle(p)$ が誤受理することがわかる. 以下の定理は, $Oracle(p = p_1p_2 \ldots p_m)$ が初めて誤受理するときの必要十分条件を述べている.

**定理 1.** 状態 $i+1$ まで構成し, 状態 $i$ まで誤って受理される文字列がないと仮定する. $SP_p(i) = \{j_0 = i, j_1, \ldots, j_{l-1}, j_l = 0\}$, $\delta(i, \sigma) = i+1$ $(p_{i+1} = \sigma)$, $u_k = \mathrm{repet}_p(j_k)$, $L(j_k)$ を状態 $j_k$ で受理される文字列の集合とすると,

$\exists k \in [1 \ldots l-1], \exists v \in L(j_k)[(|v| > |u_{k-1}|) \wedge \delta(j_k, \sigma) = i+1]$
$\Leftrightarrow \exists v \in \Sigma^*$[ 状態 i+1 で $v\sigma$ は誤って受理される.]

*Proof.* ($\Rightarrow$)

条件を満たすような文字列 $v$ を一つとってくる. ここで, 状態 $i+1$ で受理される誤受理ではない文字列 ($p$ の部分文字列) は $p_1p_2 \ldots p_{i+1}$ の suffix のみである. よって, $v \notin \mathrm{suf}(i)$ ならば $v\sigma(= vp_{i+1}) \notin \mathrm{suf}(i+1)$ となるから, $v\sigma$ は状態 $i+1$ で誤受理されることになる. 以下では, $v \notin \mathrm{suf}(i)$ を示す.

$|v| > |u_{k-1}|$ より, $p \notin \mathrm{suf}(j_{k-1})$ .

$v \in \mathrm{suf}(i)$ と仮定する. $v \in \mathrm{suf}(i)$ だから, $|u_{t+1}| < |v| \leq |u_t|$ となるような $t$ が存在する. このとき $S_p(j_{t+1}) = j_{t+2} = j_k$ となり, $v \in \mathrm{suf}(j_{t+1} = j_{k-1})$ となる. よって,$v \notin \mathrm{suf}(i)$.

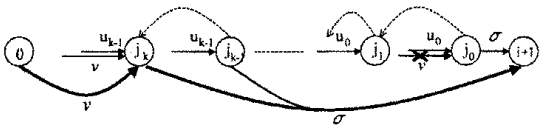


図 2: $v\sigma$ が誤って受理される時のファクターオラクル

($\Leftarrow$) $\exists v \in \Sigma^*$ に対して, $v\sigma$ が状態 i+1 で誤って受理されるということは, $v\sigma \notin \mathrm{suf}(i+1), \delta(i, \sigma) = i+1$. また, $\exists k \in [1 \ldots l-1]$, 文字列 $v$ を受理する状態として $j_k \in SP_p(i)$ なので, $v \in L(j_k)$.

今, $|v| \leq |u_{k-1}|$ とすると, $v \in \mathrm{suf}(j_k)$ でもあるから, $v$ は $u_{k-1}$ の suffix となる. すると [ACR99][Corollary4] より, $u_{k-1}$ は $u_0$ の suffix だから $v$ は $u_0$ の suffix にもなる. これは,$v \in \mathrm{suf}(i)$ となる. また, $v\sigma \notin \mathrm{suf}(i+1)$ より, $v \notin \mathrm{suf}(i)$ であり矛盾. よって,$|v| > |u_{k-1}|$ □

ファクターオラクルがはじめて誤受理するときの必要十分条件である定理 1 から, $Oracle(p)$ を構成していき条件を満たすような文字列 $v$ を見つけることを第一段階とする. その後, $v$ を受理する状態から新たに遷移が定義され, 誤受理することを第二段階とする. 第一段階をクリアしたとき, つまり条件を満たすような文字列 $v$ が見つかったときをリーチと呼ぶことにする. まずはじめに, 判定条件の一つである $\mathrm{repet}_p(i)$ の求め方について述べていく.

### 3.2 $|\mathrm{repet}_p(i)|$ の近似値の計算

[LL00] において, $|\mathrm{repet}_p(i)|$ の近似値 $lrs(i)$ の計算法が述べられている. $lrs(i)$ は

次のように再帰的に定義され, $p[1\ldots i]$ の反復接尾辞の一つ (最長の反復接尾辞 $\mathrm{repet}_p(i)$ であるとは限らない) の長さを与え, それが位置 $S_p(i)$ に出現すること, またこの定義に従ったアルゴリズムによって $Oracle(p)$ と全ての $i(1 \leq i \leq |p|)$ に対して $lrs(i)$ を $O(|p|)$ 時間で計算可能なことを証明している.
状態 $j$ を $Oracle(p[1\ldots i])$ と $p_{i+1}$ から $Oracle[1\ldots i+1]$ を構成するときに $SP_p(i)$ をたどったとき最後の状態, つまり $\delta(j, p_{i+1})$ が既に定義されていた最大の $SP_p(i)$ 上の状態とする.

**定義 4.** ([LL00] **Definition 8**).
$\pi_1 \overset{\text{def}}{\Leftrightarrow} S_p(\pi_1) = j \cap \pi_1 \in SP_p(i)$.

**定義 5.** ([LL00] **Definition 9**).

$$\pi_2 \overset{\text{def}}{\Leftrightarrow} \begin{cases} S_p(\pi_2) = j \cap \pi_2 \in SP_p(S_p[i+1]-1). \\ \qquad (S_p(i+1)-1 \neq j \text{ のとき}). \\ j (S_p(i+1)-1 = j \text{ のとき}). \end{cases}$$

**定義 6.** ([LL00] **Definition 10**).
$lrs$ : サイズ $m+1$ の整数配列, $\forall i (0 \leq i < m)$

$$lrs[i+1] = \begin{cases} 0 & (S_p(i+1) = 0 \text{ のとき}). \\ lrs[\pi_1]+1 & (\pi_2 = S_p(\pi_1) \text{ のとき}). \\ \min\{lrs[\pi_1], lrs[\pi_2]\}+1 & (\text{その他}). \end{cases}$$

$lrs[0] = 0$.

このように定義した $lrs[i]$ の値は $|\mathrm{repet}_p(i)|$ の近似値なので, 必ずしも正確な $|\mathrm{repet}_p(i)|$ の値を示すとは限らない. そこで, 定理 1 に $|\mathrm{repet}_p(i)|$ の代わりに $lrs[i]$ を適用するために次の補題を示す.

**補題 1.** $Oracle(p[1\ldots i+1])$ が誤受理していないとすると, $lrs[i+1] = |\mathrm{repet}_p(i+1)|$.

*Proof.* $i$ に関する帰納法でこれを示す. $i = 0$ のとき: $lrs[1] = 0$, $\mathrm{repet}_p(1) = \epsilon$ となるので明らか.
$i > 0$ のとき: $\forall j : 0 \leq j \leq i$, $lrs[j] = |\mathrm{repet}_p(j)|$ が成り立つと仮定する.

- $S_p(i+1) = 0$ のとき: $\mathrm{repet}_p = \epsilon$. また, 定義 6 から $lrs[i+1] = 0$. よって成立する.
- $S_p(i+1) = q (q \neq 0)$ : このケースは更に 2 つのケースに分かれる.
  - $q - 1 = \pi_2 = S_p(\pi_1)$:定義に従って示せる.
  - $q - 1 \neq S_p(\pi_1)$: 誤受理していないということは, 任意の状態 $i \in Oracle(p[1\ldots i])$ に対して, $w \in L(i)$ ならば $w \in \mathrm{suf}(i)$ である. また, 誤受理してない性質を用いると $\min\{lrs[\pi_1], lrs[\pi_2]\} = lrs[\pi_1]$ が言える. この二つを用いて, $|\mathrm{repet}_p(\pi_1)| + 1 = lrs[i] + 1 = |\mathrm{repet}_p(i+1)|$ を示せる.

□

### 3.3 誤受理判定アルゴリズム

◎ファクターオラクルの誤受理判定問題
入力: 文字列 $p = p_1 p_2 \ldots p_m$
質問: $Oracle(p)$ は誤受理するか?

補題 1 を定理 1 に用いて $Oracle(p)$ の誤受理判定アルゴリズムを構成する. まず, $Oracle(p[1\ldots i])$ と $p_{i+1}$ から $Oracle(p[1\ldots i+1])$ を構成するとき, 同時に $lrs[i+1]$ を計算し保存していくようにする. $Oracle(p[1\ldots i])$ まで構成し, はじめて, $S_p(i) > lrs[i]$ となったときがはじめてリーチがかかる時である. なぜなら, 補題 1 から $S_p(i) > |\mathrm{repet}_p(i)|$ であり, 定理 1 の文字列 $v$ として少なくとも $v = p[1\ldots S_p(i)]$ がある. よって, 条件を満たすような $v$ が存在するのでリーチとなる. リーチがかかった後は, 文字を加えるたびに外部遷移が出るかどうか調べていけばよい. 外部遷移が出れば定理 1 の条件が満たされ, その時点で $Oracle(p)$ は誤受理することになる. つまり, 一文字加える度に定理 1 の二つの条件を満たすかどうか調べていく (図 3).

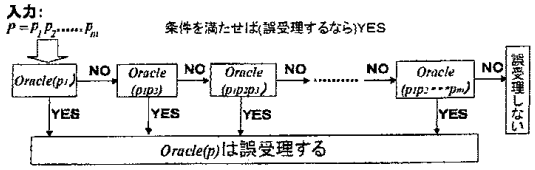


図 3: 誤受理判定アルゴリズムの概要

**NewOracle-on-line** $(p = p_1p_2\cdots p_m)$
1.Create $Oracle(\epsilon)$
2. one single state 0
3. $S_\epsilon(0) = -1$, flag $\leftarrow 0$
4.**For**(i **from** 1 **to** m)
5. $Oracle(p[1\ldots i]) \leftarrow$
6. NewAddLetter($Oracle(p[1\ldots i-1], p_i)$)
7. **if**(flag == 0)
8. **if**($S_p(i) > lrs[i]$),
9. **then** flag $\leftarrow$ 1 (リーチ).
10. **else if**(flag == 1)
11. **if**( 状態 $i$ へ外部遷移を構成 ),
12. **then** flag $\leftarrow$ 2 (誤受理).
13.**End For**

図 4: アルゴリズム NewOracle-on-line

$Oracle(p)$ が完成するまでにリーチがかからない, あるいはリーチがかかってもその後外部遷移が出なければ誤受理はしない. 具体的なアルゴリズムを図 4 に示す. 変数 flag は, 0,1,2 の値をとり, flag= 2 となると誤受理となる. 今までファクターオラクルの構成に用いてきた **AddLetter** を **NewAddLetter** に変更してファクターオラクルを構成する. **NewAddLetter** に出てくる **LRS**$(\pi_1, S_p(i+1))$ は, $lrs(i+1)$ の値を返す. また, **LCS**$(\pi_1, \pi_2)$ は, $p[1\ldots i]$ と $p[1\ldots S_p(i+1)-1]$ の共通 suffix の長さを返す. 誤受理判定を伴ったファクターオラクルの構成法は次のようになる (図 4).

**定理 2.** アルゴリズム **NewOracle-on-line**$(p = p_1p_2\cdots p_m)$ は, $Oracle(p)$ と $lrs[i](\forall i, 0 \le i \le m)$ を計算する. また,

**NewAddLetter**$(Oracle(p[1\ldots i], \sigma))$
1.Create a new state $i+1$
2.$\delta(i,\sigma) \leftarrow i+1$
3.$j \leftarrow S_p(i)$
4.$\pi_1 \leftarrow i$
5.**while** $j > -1$ and
6. $\delta(j,\sigma)$ is undefined **do**
7. $\delta(j,\sigma) \leftarrow i+1$
8. $\pi_1 \leftarrow j$
9. $j \leftarrow S_p[j]$
10.**if** $j = -1$ **then**
11. $s \leftarrow 0$
12.**else** $s \leftarrow \delta(j,\sigma)$
13.$S_p(i+1) \leftarrow s$
14.$lrs[i+1] \leftarrow$ LRS$(\pi_1, S_p(i+1))$
15.**return** $Oracle(p[1\ldots i]\sigma)$

図 5: アルゴリズム NewAddLetter

$Oracle(p)$ が誤受理するかどうか判定する.

*Proof.* $Oracle(p)$ と $lrs[i]$ の計算に関しては, [ACR99](Theorem 1) と [LL00](Lemma 5) を用いて文字列 $p$ についての帰納法で示される. 誤受理判定に関しては, 補題 1 と定理 1 を用いればいえる. □

**定理 3.** アルゴリズム **NewOracle-on-line**$(p = p_1p_2\ldots p_m)$ の複雑さは, 計算量 (時間計算量), 空間量 (空間計算量) ともに $O(m)$ である.

*Proof.* [ACR99](Theorem 2) より Oracle(p) の計算は, $O(m)$. 配列 $lrs[0\ldots m]$ の空間量は明らかに $O(m)$ なので, 計算量について示す. 定義 6 のはじめの二つのケースは定数時間で計算できる. 3 つ目のケースであるが $\pi_2$ を見つけるために suffix path$SP_p(S_p(i+1)-1)$ を何回辿るのかが問題である. ここで $Oracle(p[1\ldots S_p(i+1)])$ が構成されたときのことを考える. $k = S_p(i+1)$ とすると, 遷移 $\delta(j, p_k)$ が定義されているはずである. $Oracle(p[1\ldots S_p(i+1)])$ を構成するのに, suffix path $SP_p(S_p(i+1)-1)$

**LRS**$(\pi_1, s)$
1.**if** $s = 0$ **then**
2. **return** 0
3.**else** **return** LCS$(\pi_1, s-1)+1$

図 6: $p[1 \ldots i+1]$ の $lrs(i+1)$ を計算

**LCS** $(\pi_1, \pi_2)$
1.**if** $\pi_2 = S_p(\pi_1)$ **then**
2. **return** $lrs[\pi_1]$
3.**else** **while** $S_p(\pi_2) \neq S_p(\pi_1)$ **do**
4. $\pi_2 \leftarrow S_p(\pi_2)$
5.**return** $\min\{lrs[\pi_1], lrs[\pi_2]\}$

図 7: $p[1 \ldots i]$ and $p[1 \ldots S_p(i+1)-1]$ の共通 suffix の長さを計算

を辿った回数より, $\pi_2$ を見つけるために suffix path を辿る回数は等しいか少ない. よって, $Oracle(p)$ を構成するときに suffix path を辿る合計回数は $O(m)$ となる. 結局, $lrs[0 \ldots m]$ の計算も $O(m)$. また, 誤受理判定に関わる 6 ~ 9 行目に関しても一文字加えたときに最大で 4 回の比較計算なので $m$ 文字加えても $4m$ 回. 以上から, **NewOracle-on-line**$(p = p_1 p_2 \ldots p_m)$ の時間計算量, 空間計算量ともには, $O(m)$. □

例として, $Oracle(abbbc)$ をこのアルゴリズムで構成すると図 8 のようになる.

## 4 結論と今後の課題

ファクターオラクルが部分文字列以外を受理 (誤受理) することについて解析し, $Oracle(p)$ が初めて誤受理するときの必要十分条件を与えた. また, $Oracle(P)$ が誤受理するかどうか線形時間 ($O(|p|)$) で判定する方法を提案した.

本研究の動機としてファクターオラクルは, 構成に使われた文字列の部分文字列以外を受理することがあるという性質があった. しかし, PSC '04 においてその手掛かりとなるよ

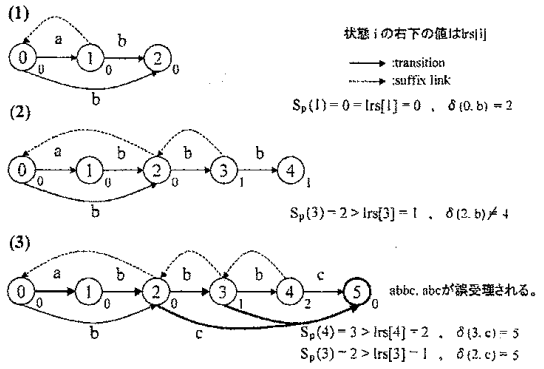


図 8: $Oracle(abbbc)$ の誤受理判定

うな特徴付けが報告されている [AC04].
本論文では初めて誤受理する場合について解析したが, 2 回目以降誤受理される文字列については分かっていない部分が多い. また, $Oracle(p)$ は状態数 $|p|+1$ 個で $p$ の全ての部分文字列を受理するオートマトンではあるが, 最小のものではないという未解決問題も残っている. 言い換えると, 外部遷移の数を $Oracle(p)$ よりも少なくできるオートマトンが存在し, その構成法がまだよくわかっていない.

## 参考文献

[AC04] Alban Mancheron and Christophe Moan. Combinatorial Characterization of the Language Recognized by Factor and Suffix Oracles In *Procedings of the Prague Stringology Conference '04*, pp. 139-153, 2004.

[ACR99] M. Allauzen, M. Crochemore, and M. Raffinot. Factor oracle: a new structure for pattern matching. *Theory and Practice of Informatics* number 1725, 291-306, 1999.

[LL00] A. Lefebvre, T. Lecroq. Computing repeated factors with a factor oracle, *Proc. of the 11th Australasian Worksho on Combinatorial Algorithms* page339-348, 2000.

[McC76] Edward M. McCreight. A space-economical suffix tree constrction algorithm, *Journal of the ACM* 23:262-272, 1976.

[MM90] U. Manber, G. Myers. A new method for on-line string searches, *1st Annual ACM-SIAM Symposium on Discrete Algorithms* page319-327, 1990.