複数パターンマッチングマシンにおける遷移先決定関数の最適化

蔵満 琢麻† 松浦 寛生† 望月 久稔†
†大阪教育大学

1. はじめに

コンピュータの処理において複数パターンの照合は様々な場面で利用され，高速性が求められる．Aho-Corasick 法 [1] は，マシン AC と呼ばれる照合機械を用いて，複数パターンを線形時間で照合できる手法である．マシン AC における Failure 関数による遷移ではテキストの照合ポインタは進まない．よってマシン AC における Failure 遷移を抑制することで照合時間を短縮することが考えられる．しかし，すべての遷移種に対応した遷移先を保持する領域を各状態に設けると，照合時に必要な空間が膨大になる．本論文では，Goto 関数による遷移をもたない状態において，Failure 遷移を抑制することで照合速度を高速化する手法を提案する．

2. ダブル配列によるマシン AC

マシン AC は，Goto 関数，Failure 関数，Output 関数から構成され，ダブル配列 [2][3] を用いた効率的な格納手法 [4] が提案されている．この手法は，ダブル配列のデータ構造である配列 Base, Check を用いてマシン AC を実装する．以下，状態 x における Base 値，Check 値をそれぞれ $B[x]$, $C[x]$ とする．

2.1 Goto 関数

ダブル配列によるマシン AC[4] において，状態 s から遷移種 a による状態 t への Goto 遷移は式 (1) で定義される．ある状態 s で Goto 関数が呼び出されたとき，式 (1) により状態 t への遷移の有無を確認し，遷移が存在すれば状態 t へ Goto 遷移する．遷移が存在せず，状態 s が初期状態であればテキストの照合ポインタを 1 つ進めることで初期状態自身へ Goto 遷移し，初期状態でなければ Failure 関数が呼び出される．

$$\begin{cases} t = B[s] + a \\ C[t] = a \end{cases} \quad (1)$$

2.2 Failure 関数

Failure 関数は初期状態へ遷移するものを除いてダブル配列における状態として定義される．'$' をキーに使用しない遷移種として定義し，式 (2) で示す状態 s_f の Base 値に，状態 s から Failure 関数により遷移すべき状態の Base 値を写像することで Failure 関数は定義される．ある状態 s で Failure 関数が呼び出されたとき，式 (2) により s から s_f への遷移の有無を判定し，遷移が存在すれば状態 s_f へ Failure 遷移し，存在しなければ初期状態へ Failure 遷移する．

$$\begin{cases} s_f = B[s] + \text{'\$'} \\ C[s_f] = \text{'\$'} \end{cases} \quad (2)$$

2.3 Output 関数

Failure 関数と同様に，Output 関数もダブル配列における状態として定義される．'#' をキーに使用しない '$' と異なる遷移種

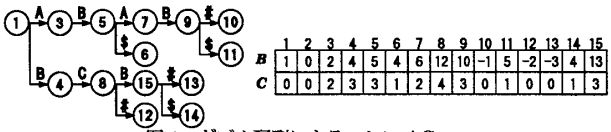

図 1 ダブル配列によるマシン AC

として定義する．式 (3) で示す状態 s_o の Base 値に，状態 s における Output 集合のインデクスを負値で格納することで Output 関数は定義される．以下，状態 n における Output 集合のインデクスを保持している状態を O_n とする．照合時，Goto 遷移により遷移した状態を s とすると，式 (3) により O_s の有無を判定し，O_s が存在すれば O_s の Base 値の絶対値を取り出すことで状態 s がもつ Output 集合が検出される．

$$\begin{cases} s_o = B[s] + \text{'\#'} \\ B[s_o] = -(s \text{ における Output 集合のインデクス}) \\ C[s_o] = \text{'\#'} \end{cases} \quad (3)$$

例 1: 図 1 にキー集合 K\{"ABAB", "BC", "BCB"\} を登録したマシン AC の状態遷移図と，配列に格納した値を示す．ここで，Output 集合 {"ABAB"}, {"BC"}, {"BCB"} のインデクスをそれぞれ 1, 2, 3 とし，遷移種 '#', '$', 'A', 'B', 'C' の内部表現値を 0, 1, 2, 3, 4 とする．また，状態遷移図における実線は Goto 遷移を表している．図 1 のマシン AC に対して，テキスト "ABABC" を入力した場合について考える．まず初期状態 1 において，遷移種 'A' による Goto 遷移が定義されているため，状態 3 へ Goto 遷移する．状態 3 において O_3 への遷移が存在しないため，ここで Output 集合は検出されない．同様に 'B', 'A', 'B' と遷移を繰り返し，状態 9 へ遷移する．ここで，状態 9 において O_9 への遷移が存在するため，$-B[B[9] + \text{'\#'}] = 1$ に該当する Output 集合 {"ABAB"} が検出される．また，状態 9 においては遷移種 'C' による Goto 遷移が定義されていないため，Failure 関数を呼び出し，状態 11($B[9]$ + '$') へ Failure 遷移する．しかし，状態 11 においても遷移種 'C' による Goto 遷移が定義されていないため，再び Failure 関数を呼び出し，状態 6 へ Failure 遷移する．その後，状態 8 へ Goto 遷移を行い，Output 集合 {"BC"} を検出して終了する．

3. 遷移先決定関数の最適化

3.1 Failure 遷移の抑制

例 1 の状態 9 のように，Goto 関数による遷移が存在しない状態をマシン AC の葉と定義する．葉においては，Goto 関数による遷移が存在しないため，確実に Failure 遷移を行うことになる．そこで，葉の Base 値に Failure 遷移先の状態の Base 値を写像することで，葉を Failure 遷移した後の状態として擬似的に取り扱う．これにより葉における Failure 遷移が抑制でき，照合速度の高速化を図れる．

例 2: 図 2 にキー集合 K を登録した最適化後のマシン AC の状態遷移図と，配列に格納した値を示す．ここで，遷移種 'A', 'B', 'C', '$', '#', の内部表現値を 0, 1, 2, 3, 4 とし，Output 集合のインデクスは例 1 と同様にする．図 2 のマシン AC における状態 9 は葉に該当するノードであるため，状態 9 の Base 値に

The Optimization of The Transition Function in Plural Patterns Matching Machine
† Takuma KURAMITSU, Nobutaka MATSUURA, Hisatoshi MOCHIZUKI Osaka Kyoiku University

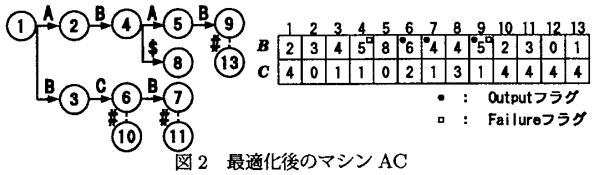

図 2　最適化後のマシン AC

Failure 遷移先の状態 4 における Base 値 5 を格納する．これにより，Failure 関数を定義する状態 (図 1 のマシン AC における状態 11 に該当する状態) を抑制できる．また，状態 9 は状態 4 における遷移をもつため，照合時の Failure 遷移回数を抑制することができる．

しかし，ダブル配列によるマシン AC[4] において，マシン AC の葉における Base 値は Output 集合を管理する状態と関連性があるため，Output 集合の管理方法を変更する必要がある．

3.2　Output 集合の管理方法

ある状態 s における Output 集合のインデクスを管理する状態を s_o とし，遷移種の最大値を a_{max} とする．'#' $= a_{max} + 1$ と定義し，状態 s_o を式 (4) で定義する．式 (4) により状態 s_o を定義することで状態 s の Base 値と状態 s_o との関連性を取り除くことができる．

$$\begin{cases} s_o = s + \text{'\#'} \\ B[s_o] = s \text{ における Output 集合のインデクス} \\ C[s_o] = \text{'\#'} \end{cases} \quad (4)$$

3.3　フラグを用いた遷移先確認

ある状態 s において，Failure 関数を定義する状態 s_f と Output 関数を定義する状態 s_o への遷移の有無を確認するためにフラグを用いる．Base 値を管理する変数の最上位ビットを状態 s_o への遷移の有無を判定する Output フラグとして使用し，上位 2 ビット目を状態 s_f への遷移の有無を判定する Failure フラグとして使用する．

例 3: 図 2 のマシン AC を用いて，例 1 と同様に，テキスト "ABABC" を入力した場合について考える．まず初期状態 1 において，遷移種 'A' による Goto 遷移が定義されているため，状態 2 へ Goto 遷移する．状態 2 における Output フラグは立っていないため，ここで Output 集合を検出しない．同様に 'B'，'A'，'B' と遷移を繰り返し，状態 9 へ遷移する．ここで，状態 9 において，Output フラグが立っているため，$B[9 + \text{'\#'}] = 1$ に該当する Output 集合 {"ABAB"} を検出する．また，状態 9 においては遷移種 'C' による Goto 遷移が定義されておらず，状態 9 における Failure フラグが立っているため，状態 $8(B[9] + \text{'\$'})$ へ Failure 遷移する．その後，状態 6 へ Goto 遷移し，Output 集合 {"BC"} を検出して終了する．例 1 と比較すると，マシン AC の葉における Failure 遷移を抑制したことにより状態数が 2 つ少なく，また，照合時の Failure 遷移が 1 回少ないことが分かる．

4．　実験による評価

最適化による有効性を示すため，最適化を行う前のダブル配列によるマシン AC との比較実験を Intel Pentium Dual 1.60GHz, Fedora 7 上で行った．実験では，ランダムに抽出した英単語 30 万語 (以下キー集合 U とする) を用いた．また，遷移のラベルとなるキーの各構成文字の内部表現値は ASCII コードとした．

キー集合 U を母集団として 1 万語から 30 万語まで 1 万語毎に単語数を増やしたキー集合と，U を連結してテキストサイズを 10MB に調節したテキストを作成し，各キー集合とテキストを照合した．それぞれの実験において要した照合時間を図 3 に示す．また，表 1 に，今回の実験において，キー集合 U をマシン AC に登録したときの状態数，照合時に必要な空間，照合時の遷移回数，照合時間を示す．

表 1　登録キー数 30 万件における性能

	最適化前	最適化後
Failure 遷移回数 (回)	4,361,005	3,391,834
総遷移回数 (回)	14,361,005	13,391,834
状態数 (個)	2,629,593	2,394,991
照合時間 (sec)	1.922	1.656
照合時に必要な空間 (MB)	15.778	14.370

図 3 に示すように，最適化前と比べて最適化後の高速性が増していることが分かる．これは表 1 に示すように，Failure 遷移の抑制により総遷移回数が減少したためである．

また，表 1 より，最適化前と比較して照合時に必要な空間が減少していることが分かる．これは，マシン AC の葉における遷移先決定関数の最適化により，Failure 関数を定義する状態が減少したためである．

以上より，マシン AC の遷移先決定関数を最適化することにより，照合時間を短縮し，照合時に必要な空間を抑制できることを示した．

5．　おわりに

本論文では，マシン AC における遷移先関数の最適化により Failure 遷移と状態数を抑制することで，照合速度の高速化と照合時に必要な空間の抑制ができることを示した．今後の課題として，構築を含めた評価が挙げられる．

参考文献

[1] Aho A.V., Corasick M.J., Efficient String Matching An Aid to Bibliographic Search, Comm. ACM, vol.18, No.6, pp.333-340, 1975.

[2] J. Aoe, An Efficient Digital Search Algorithm by Using a Double-Array Structure, IEEE Trans. on Software Engineering, Vol.15, No.9, pp.1066-1077, 1989.

[3] 矢田 晋，森田 和宏，泓田 正雄，平石 亘，青江 順一，ダブル配列におけるキャッシュの効率化. FIT2006, pp.71-72, 2006.

[4] 信種 真人，森田 和宏，泓田 正雄，青江 順一，ダブル配列を用いたマシン AC の効率的格納方法，言語処理学会第 13 回年次大会，pp.831-834, 2007.

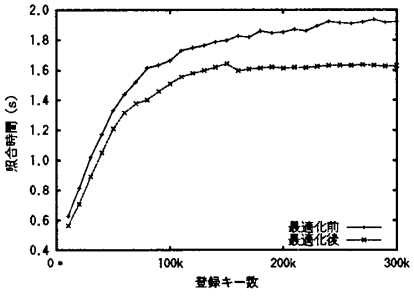

図 3　照合時間