# 2차원 사전 정합을 위한 실용적인 알고리즘

이 광 수†


## 요 약

2차원 사전 정합 문제에서는 2차원 텍스트 $T$와 2차원 패턴의 집합인 사전 $D=\{P_1, ..., P_k\}$이 주어질 때 $T$에 나타나는 모든 패턴들의 위치를 구하는 것을 목적으로 한다. 본 논문에서는 한 개의 패턴만 처리할 수 있는 새로운 2차원 패턴 정합 알고리즘을 제안하고, 이를 2차원 사전 정합 알고리즘으로 확장하는 방법을 제시한다. 제시된 2차원 사전 정합 알고리즘은 패턴 사전에 포함된 패턴의 크기나 모양이 다양한 경우에도 적용될 수 있으며, 실제 적용에서 텍스트 문자들의 일부만 검사하게 되어 수행 시간이 단축되고, 추가적으로 필요한 메모리도 사전의 크기에 비례하는 만큼만 사용하며, 특히 복잡한 자료구조를 사용하지 않음으로써 구현이 단순한 실용적인 알고리즘이다.


# A Practical Algorithm for Two-Dimensional Dictionary Matching

Gwang-Soo Rhee†


## ABSTRACT

In two-dimensional dictionary matching problem, we are given a two-dimensional text $T$ and a dictionary $D=\{P_1, ..., P_k\}$ as a set of two-dimensional patterns. We seek the locations of all the dictionary patterns that appear in $T$. We present a new two-dimensional pattern matching algorithm that can handle just a single pattern, and then show how to extend it into two-dimensional dictionary matching algorithm. The suggested algorithm is practical in the sense that it can deal with patterns of various sizes and shapes, that it runs fast in practice examining only a small fraction of the text characters, that it uses a small extra space proportional to the size of the dictionary, and that it is quite simple to be implemented without depending on complicated data structures.


## 1. 서 론

고전적인 문자열 정합 문제는 입력 자료로 길이가 $n$인 텍스트 문자열 $T$와 길이가 $m$인 패턴 문자열 $P$가 주어지면 텍스트 상에서 패턴이 나타나는 모든 위치, 즉 $T[i..(i+m-1)]=P$를 만족하는 모든 $i$들을 결과로 출력하는 문제이다. 이 문제는 심도 있게 연구되어 왔으며, Knuth와 Morris와 Pratt 등이 최초의 선형 시간 알고리즘을 제시하였다 [13]. 비슷한 시기에 Boyer와 Moore가 실용적인 면에서 보다 개선된 알고리즘을 발표하였다 [9]. Karp와 Rabin은 해쉬 함수를 사용하여 평균적으로 선형 수행 시간을 요하는 알고리즘을 발표하였다 [12].

사전 정합 문제는 문자열 정합 문제를 일반화한 것으로 그 입력 자료는 텍스트 문자열 $T$와 패턴 문자열 집합 $D=\{P_1, ..., P_k\}$이며, 출력 결과는 $T[i..(i+m-1)]=P_j$를 만족하는 모든 순서쌍 $(i, j)$들이다. 패턴 문자열들의 길이가 일정하지 않을 경우 $j$번째 패턴의 길이를 $m_j$라고 하면 패턴 출현 조건식은 $T[i..(i+m_j-1)]=P_j$가 된다. 사전 정합 알고리즘으로는 Aho와 Corasick이 유한 오토마타를 응용한 알고리즘을 발표하였다 [1]. 사전 정합을 보다 일반화한 것으로 동적 사전 정합 문제가

† 종신회원 : 숙명여자대학교 전산학과 교수
논문접수 : 1998년 8월 5일, 심사완료 : 1999년 1월 16일

있는데, 여기서는 패턴 집합 $D$에 패턴의 추가와 삭제를 허용한다. Meyer는 패턴의 추가만을 허용하는 알고리즘을 발표하였으며 [14], Amir 등은 패턴의 추가와 삭제를 모두 허용하는 효율적인 알고리즘을 발표하였다 [4].

2차원 패턴 정합 문제에서는 패턴과 텍스트가 직사각형 형태로 주어지는 것으로 주로 영상처리와 관련하여 제기되었다. 입력 자료는 직사각형 텍스트 $T[1..n_1, 1..n_2]$와 직사각형 패턴 $P[1..m_1, 1..m_2]$이며, 출력 결과는 $T[i_1..(i_1+m_1-1), i_2..(i_2+m_2-1)]=P$를 만족하는 텍스트 상의 모든 위치들 $(i_1, i_2)$이다. 최초의 선형 시간 알고리즘은 Baker와 Bird가 각각 독립적으로 발견하였는데 [7,8], 사용된 알파벳의 크기 $\sigma$가 상수로 취급되지 않을 경우 그 수행 시간은 $O(n_1 n_2 \log \sigma)$로 주어진다. Amir 등은 알파벳의 크기와 관계없이 수행 시간이 $O(n_1 n_2)$으로 주어지는 이론적으로 효율적인 알고리즘을 발견하였다 [2]. 2차원 패턴 정합의 경우 특히 패턴과 텍스트가 크기 때문에 이론적으로보다는 실제 응용에서 빨리 수행되는 알고리즘의 설계는 보다 큰 의미를 갖는다. Zhu와 Takaoka는 Karp-Rabin 알고리즘과 Knuth-Morris-Pratt 알고리즘, Karp-Rabin 알고리즘과 Boyer-Moore 알고리즘을 혼합한 두 개의 알고리즘을 제시하고, Baker와 Bird가 발표한 알고리즘보다 빠름을 시뮬레이션 결과를 사용하여 주장하였다 [16]. Baeza-Yates와 Régnier는 실제 응용에서 텍스트의 일부만 검사하는 보다 개선된 알고리즘을 발표하였는데, 자료에 따라서는 텍스트의 $1/m_1$ 정도만 검사하는 경우도 있다 [6]. 그들의 알고리즘에서는 텍스트의 매 $m_1$번째 줄을 검사하여 패턴의 어느 한 줄과라도 일치하는 부분이 발견되면 그 위치에서 전체 패턴과의 비교를 통해 정합 여부를 조사하는 방법을 사용하며, 패턴이 어느 정도 커지면 우연에 의한 패턴 한 줄과 텍스트 부분의 일치 가능성이 작으므로 전체 패턴이 비교되는 경우가 많지 않게 되고 수행 시간의 기대치가 $O(n_1 n_2/m_1)$에 접근함을 주장하였다. 그러나, 텍스트와 패턴의 줄 단위의 비교 과정에서 사용되는 유한 오토마타가 패턴의 크기에 비례하여 복잡해지므로, 패턴의 크기 증가에 의한 수행 시간 감소 효과가 $1/m_1$에 비례하지는 않는다.

2차원 사전 정합 문제는 2차원 패턴 정합의 일반화이며, 입력 자료에 하나의 직사각형 패턴 대신에 직사각형 패턴 집합이 사용되며, 패턴들의 크기는 각기 다를 수도 있다. Amir와 Farach는 패턴들이 모두 정사각형인 경우 이론상으로 효율적인 알고리즘을 발견하였으며, 그 수행 시간은 $O((m+n)\log k)$인데, 여기서 $k$는 패턴의 수, $m$은 패턴들의 크기의 합, $n$은 텍스트의 크기이다 [3]. 정사각형 패턴들은 한 귀를 정하여 정돈시킬 수 있으며, Amir와 Farach는 이 점을 이용하여 텍스트의 대각선 방향으로 패턴과 쉽게 비교할 수 있는 방법을 고안하고, 사전 정합은 Aho-Corasick 알고리즘을 응용하였다. Amir-Farach 알고리즘이 갖는 의미는 사전 정합 문제의 해결 시간이 최악의 경우에 대한 수행 시간을 기준으로 할 때 패턴의 수 $k$가 아니라 $\log k$에 비례하여 증가하는 방법이 발견되었다는 것이다. Idury와 Schäffer는 패턴의 형태에 제한을 두지 않는 보다 일반화된 알고리즘을 발표하였으나 [11], 그 수행 시간은 정사각형 패턴의 경우 Amir-Farach 알고리즘에 비해 현저히 느리다.

2차원 패턴 정합 문제를 위한 실용적인 알고리즘인 Zhu-Takaoka 알고리즘이나 Baeza-Yates와 Régnier의 알고리즘은 2차원 사전 정합 문제로의 확장이 연구되어 있지 않으며, 특히 패턴들의 크기가 일정하지 않을 때는 각 패턴에 대하여 패턴 정합 알고리즘을 반복 수행하는 것 이외의 다른 방법이 없어 보인다. [3, 11]에서 제안된 2차원 사전 정합 알고리즘들은 실제 구현되어 사용될 수 있는 실용적인 알고리즘으로써가 아니라 최악의 경우에 대한 이론적인 수행 시간의 향상을 목적으로 하였으며, 특히 Amir-Farach 알고리즘은 정사각형 패턴이 아닌 경우에는 적용이 불가능하다. 이 논문에서는 다양한 모양과 크기의 패턴들에 대하여 적용될 수 있으면서도 구현이 용이하며 실질적인 기대 수행 시간에 있어 효율적인 알고리즘을 제시하고자 한다.

이 논문의 나머지는 다음과 같이 구성되어 있다. 2절에서는 2차원 패턴 정합을 위한 알고리즘 골격과 이를 일반화하여 2차원 사전 정합을 위한 알고리즘 골격을 기술한다. 3절에서는 앞에서 제시된 알고리즘들의 성능과 기대 성능을 향상시키기 위한 구현 방법을 제시하고 기대 성능에 대한 이론적 및 실험적 분석 결과를 토의한다. 4절에서는 제시된 알고리즘들의 특성과 확장 방법을 제시하면서 결론을 맺는다.

## 2. 기본 알고리즘

여기서 먼저 몇 가지 용어와 기호를 정의하고, 단일

직사각형 패턴에 대한 정합 알고리즘을 제시하며, 다음에 패턴 집합의 경우에 대하여 알고리즘을 일반화하는 방법을 보인다. 그리고, 이 절 이후에서는 직사각형 대신에 **사각형**이라는 표현을 같은 의미로 사용한다.

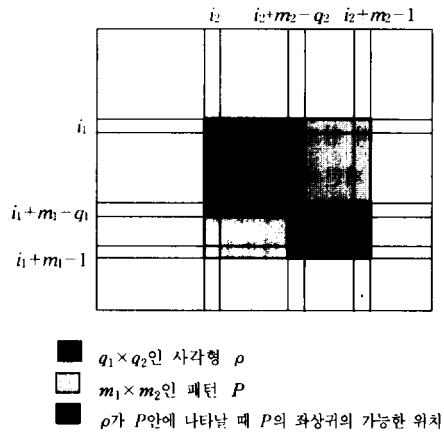
$R[1..r_1, 1..r_2]$를 주어진 사각형이라고 하자. $1 \le top \le top + rows - 1 \le r_1$과 $1 \le left \le left + cols - 1 \le r_2$를 만족하는 *top*, *left*, *rows*, *cols*에 대하여 $\rho(R, top, left, rows, cols) = R[top..(left + cols - 1)]$는 사각형 $R$의 **부분사각형**이라고 부른다. 보다 구체적으로는 $\rho(R, top, left, rows, cols)$를 $R$의 $(top, left)$ 위치에 자리하고 크기가 $rows \times cols$인 $R$의 부분사각형이라고 한다. 그리고, $\Xi(R, s_1, s_2)$는 사각형 $R$안에 자리하는 크기가 $s_1 \times s_2$인 모든 부분사각형들의 집합, 즉 $\{\rho(R, i_1, i_2, s_1, s_2) \mid (1 \le i_1 \le r_1 - s_1 + 1) \wedge (1 \le i_2 \le r_2 - s_2 + 1)\}$를 나타낸다.

사각형 $T[1..n_1, 1..n_2]$과 $P[1..m_1, 1..m_2]$를 각각 텍스트와 패턴이라고 하자. $\Xi(P, q_1, q_2)$는 패턴 $P$ 안에 들어 있는 크기가 $q_1 \times q_2$인 모든 사각형들의 집합을 나타낸다.

**[정리 1]** 사각형 $\rho(T, (i_1 + m_1 - q_1), (i_2 + m_2 - q_2), q_1, q_2)$가 집합 $\Xi(P, q_1, q_2)$에 속하지 않으면, 패턴 $P$도 집합 $\Xi(\rho(T, i_1, i_2, 2m_1 - q_1, 2m_2 - q_2), m_1, m_2)$에 속하지 않는다.

**<증명>** 위의 정리의 증명을 위해 귀류법을 사용한다. 먼저 패턴 사각형 $P$가 집합 $\Xi(\rho(T, i_1, i_2, 2m_1 - q_1, 2m_2 - q_2), m_1, m_2)$에 속한다고 가정해 보자. 그러면, $P = \rho(T, j_1, j_2, m_1, m_2)$와 $i_1 \le j_1 \le i_1 + m_1 - q_1$와 $i_2 \le j_2 \le i_2 + m_2 - q_2$를 만족하는 $j_1, j_2$가 있게 된다. $q_1 \times q_2$ 크기의 사각형 $\rho(T, (i_1 + m_1 - q_1), (i_2 + m_2 - q_2), q_1, q_2)$는 명백히 사각형 $\rho(T, j_1, j_2, m_1, m_2)$ 안에 나타나며, 따라서 집합 $\Xi(\rho(T, i_1, i_2, 2m_1 - q_1, 2m_2 - q_2), m_1, m_2)$에도 속하게 된다. 집합 $\Xi(\rho(T, i_1, i_2, 2m_1 - q_1, 2m_2 - q_2), m_1, m_2)$는 집합 $\Xi(P, q_1, q_2)$과 같으며, 정리의 전제에서 사각형 $\rho(T, (i_1 + m_1 - q_1), (i_2 + m_2 - q_2), q_1, q_2)$는 집합 $\Xi(P, q_1, q_2)$에 나타날 수 없으므로 증명 초반부의 가정이 거짓이 된다. <증명 끝>

정리 1의 의미는 다음과 같다. 만일 크기가 $q_1 \times q_2$인 사각형 $\rho(T, (i_1 + m_1 - q_1), (i_2 + m_2 - q_2), q_1, q_2)$이 패턴 $P$ 안에 나타난다면 $P$가 위치할 수 있는 곳은 (그림 1)에서 볼 수 있는 것처럼 격자무늬로 표시된 텍스트 상의 위치들 $(i_1..(i_1 + m_1 - q_1), i_2..(i_2 + m_2 - q_2))$이다. 따라서, $\rho(T, (i_1 + m_1 - q_1), (i_2 + m_2 - q_2), q_1, q_2)$이 패턴 $P$ 안에 나타나지 않는다면, 텍스트 상의 위치들 $(i_1..(i_1 + m_1 - q_1), i_2..(i_2 + m_2 - q_2))$은 패턴 $P$가 나타날 수 있는 위치에서 제외될 수 있다. 이 논문에서 제안되는 알고리즘은 앞의 사실을 이용하여 $i_1 = (1 + m_1 - q_1), 2(1 + m_1 - q_1), 3(1 + m_1 - q_1), \ldots, i_2 = (1 + m_2 - q_2), 2(1 + m_2 - q_2), 3(1 + m_2 - q_2), \ldots$와 같은 $i_1, i_2$ 값들에 대해서만 부분사각형 $\rho(T, i_1, i_2, q_1, q_2)$이 패턴 $P$ 안에 나타나는지 검사하고, 그 부분사각형이 나타나는 경우에만 패턴 $P$ 전체와 텍스트의 정합 가능 영역을 비교하는 전략을 사용함으로써 빠른 시간 안에 텍스트 영역을 검사해 나가게 된다. 그리고, 앞에서 선정된 $i_1, i_2$ 값들은 패턴이 나타날 모든 가능한 위치들을 빠짐없이 조사할 수 있도록 정해진 것이다.




(그림 1) $q_1 \times q_2$인 사각형 $\rho$를 포함하는 패턴의 위치
(Fig. 1) Positions for patterns containing $q_1 \times q_2$ rectangle $\rho$

알고리즘에서 텍스트의 부분사각형들이 패턴 안에 나타나는지를 조사하는 작업은 **텍스트 탐색 단계**라고 부

르고, 텍스트의 패턴 크기만큼의 영역과 패턴 전체를 비교하는 작업은 **텍스트 점검 단계**라고 부른다. 아래의 알고리즘에서 텍스트 탐색 단계는 기본적으로 Boyer-Moore의 문자열 정합 알고리즘을 [10]에서 개량한 Horspool 알고리즘의 2차원 판이라고 할 수 있다.

**[알고리즘 1]** 이 알고리즘은 크기가 $n_1 \times n_2$인 텍스트 $T$안에 크기가 $m_1 \times m_2$인 패턴 $P$가 나타나는 모든 위치들을, 크기가 $q_1 \times q_2$인 부분사각형들을 텍스트 탐색 단계에서 사용하여 찾는다. 알고리즘에서 사용되는 *Occurrence*$(s, P)$는 부분사각형 $s$가 나타나는 패턴 $P$ 안의 모든 위치들의 집합 $\{(j_1, j_2) | \rho(P, j_1, j_2, q_1, q_2) = s\}$를 결과 값으로 계산하는 함수이다.

```
TwoDimMatch( T, n1, n2, P, m1, m2, q1, q2) {
  for ( i1←1 + m1 − q1; i1 ≤ n1 − q1 + 1; i1←i1 + 1 + m1 − q1)
    for ( i2←1 + m2 − q2; i2 ≤ n2 − q2 + 1;
          i2←i2 + 1 + m2 − q2)
      for each ( j1, j2) in
                Occurrence(ρ( T, i1, i2, q1, q2), P)
        if ( ρ( T, i1 − j1 + 1, i2 − j2 + 1, m1, m2) = P)
          print (match at a position
                ( i1 − j1 + 1, i2 − j2 + 1))
}
```

다음에 알고리즘 1의 2차원 패턴 정합 알고리즘을 2차원 패턴 사전을 위한 알고리즘으로 일반화시킨다.

**[알고리즘 2]** $D = \{P_1, P_2, \ldots, P_k\}$는 패턴 사전이고, 각 패턴 $P_j$의 크기는 $m_1^j \times m_2^j$이다. $m_1 = \min\{m_1^1, m_1^2, \ldots, m_1^k\}$, $m_2 = \min\{m_2^1, m_2^2, \ldots, m_2^k\}$라고 하자. $q_1$, $q_2$는 $1 \le q_1 \le m_1$와 $1 \le q_2 \le m_2$를 만족하는 값으로 선정한다. 이 알고리즘은 크기가 $n_1 \times n_2$인 텍스트 $T$안에 1에서 $k$ 사이의 각 $j$에 대하여 크기가 $m_1^j \times m_2^j$인 패턴 $P_j$가 나타나는 모든 위치들을, 크기가 $q_1 \times q_2$인 부분사각형들을 텍스트 탐색 단계에서 사용하여 찾는다. 알고리즘에서 사용되는 *Occurrence*$(s, D)$는 부분사각형 $s$가 나타나는 패턴 $P_j$안의 모든 위치들과 패턴 번호의 집합 $\{(j_1, j_2, j) | \rho(P_j, j_1, j_2, q_1, q_2) = s\}$를 결과 값으로 계산하는 함수이다.

```
TwoDimMatch( T, n1, n2, P1, P2, ..., m1^1, m2^1, m1^2, m2^2,
             ..., q1, q2){
  for ( i1←1 + m1 − q1; i1 ≤ n1 − q1 + 1; i1←i1 + 1 + m1 − q1)
    for ( i2←1 + m2 − q2; i2 ≤ n2 − q2 + 1;
          i2←i2 + 1 + m2 − q2)
      for each ( j1, j2, j) in
                Occurrence(ρ( T, i1, i2, q1, q2), P1, P2, ...})
        if ( ρ( T, i1 − j1 + 1, i2 − j2 + 1, m1^j, m2^j) = Pj)
          print (match at a position
                ( i1 − j1 + 1, i2 − j2 + 1) with j-th pattern)
}
```

## 3. 구현과 성능

$\Sigma$는 텍스트와 패턴을 추출하는 알파벳을 나타내고, $\sigma = |\Sigma|$는 알파벳의 크기를 나타낸다. 이 절의 알고리즘 성능 분석에서는 불필요하게 식이 복잡해지는 것을 피하기 위해 텍스트와 패턴이 정사각형 형태임을 가정하며, 이 경우 텍스트 크기에서는 $n = n_1 = n_2$, 패턴 크기에서는 $m = m_1 = m_2$, *Occurrence*() 함수에서 사용되는 부분사각형의 크기에서는 $q = q_1 = q_2$가 사용되며, 이 가정은 **정사각형 가정**이라 부른다. 그러나 알고리즘의 성능 자체는 텍스트나 패턴의 형태와는 무관하다.

3.1에서는 먼저 제시된 알고리즘들의 수행 시간을 결정하는 주된 요소인 검사되는 문자수에 대한 기대값을 분석한다. 3.2에서는 텍스트 탐색 단계에 대한 해쉬테이블을 사용하는 효율적인 구현 방법을 제시하고, 알고리즘들의 수행시간이 검사되는 문자수에 비례함을 설명한다. 3.3절에서는 실용적인 2차원 패턴 정합 알고리즘들에 대하여 실험적으로 수행 시간을 비교하며, 또한 이 논문에서 제안된 2차원 사전 정합 알고리즘에 대한 사전의 크기가 미치는 영향을 분석한다.

### 3.1 알고리즘들에 대한 일반적 분석

앞 절의 알고리즘들에서 텍스트 문자를 검사하는 곳은 텍스트 탐색 단계의 *Occurrence*() 함수와 텍스트 점검 단계에서 전체 패턴과 텍스트 영역을 비교하는 곳이다. *Occurrence*() 함수는 정확히 $\lceil (n_1 - m_1 + 1)/(m_1 - q_1 + 1) \rceil$

· $\lceil (n_2-m_2+1)/(m_2-q_2+1) \rceil$번 호출되며, 매번 호출될 때마다 $q_1 \times q_2$ 개의 텍스트 문자를 검사한다. 따라서, 텍스트 탐색 단계에서 검사되는 텍스트 문자의 개수 $t_{scn}(n_1, n_2, m_1, m_2, q_1, q_2)$는 다음 식에 의해 주어진다.

$$t_{scn}(n_1, n_2, m_1, m_2, q_1, q_2) = q_1 q_2 \left\lceil \frac{n_1-m_1+1}{m_1-q_1+1} \right\rceil \left\lceil \frac{n_2-m_2+1}{m_2-q_2+1} \right\rceil$$

임의로 생성된 크기가 $q_1 \times q_2$인 두 개의 사각형이 정확히 일치할 확률은 $1/\sigma^{q_1 q_2}$이며, 패턴 안에는 $q_1 \times q_2$ 크기의 부분사각형이 자리할 수 있는 위치는 모두 $(m_1-q_1+1)(m_2-q_2+1)$ 개이다. 따라서, 알고리즘 1에서 매번 *Occurrence*() 함수가 출력하는 위치의 개수에 대한 기대값은 $(m_1-q_1+1)(m_2-q_2+1)/\sigma^{q_1 q_2}$이다. 그리고, 임의로 생성된 크기가 $m_1 \times m_2$인 두 개의 사각형을 비교할 때 첫 번째 일치하지 않는 문자가 나타날 때 비교를 중단하는 방법을 사용하면 비교되는 문자수의 기대값은 $(\sigma/(\sigma-1)) \times (1-1/\sigma^{q_1 q_2})$임이 알려져 있다 [5]. 그리하여, 텍스트 점검 단계에서 검사되는 텍스트 문자수의 기대값은 다음과 같이 주어진다.

$$t_{chk}(n_1, n_2, m_1, m_2, q_1, q_2) = \frac{(m_1-q_1+1)(m_2-q_2+1)}{\sigma^{q_1 q_2}} \frac{\sigma}{\sigma-1}(1-\sigma^{-q_1 q_2}) \left\lceil \frac{n_1-m_1+1}{m_1-q_1+1} \right\rceil \left\lceil \frac{n_2-m_2+1}{m_2-q_2+1} \right\rceil$$

정사각형 가정 아래 전체 검사되는 텍스트 문자수의 기대값은 다음과 같이 주어진다.

$$t(n, m, q) = t_{scn}(n, m, q) + t_{chk}(n, m, q) = \left(q^2 + \frac{(m-q+1)^2(\sigma^{q^2}-1)}{(\sigma-1)\sigma^{2q^2-1}}\right) \left\lceil \frac{n-m+1}{m-q+1} \right\rceil^2$$

대부분의 실제 응용에서는 $t_{chk}(n, m, q)$ 항의 분모에 들어 있는 $\sigma^{2q^2-1}$의 값이 크므로 $t(n, m, q)$의 값에서 큰 비중을 차지하는 것은 $t_{scn}(n, m, q)$이며, 그 값은 대략 텍스트 크기의 $q^2/m^2$이다. <표 1>의 $k=1$이 나타내는 열은 몇몇 $\sigma$와 $q$ 값에 대해 $t_{scn}(n, m, q)$ 값이 $t_{chk}(n, m, q)$ 값보다 큰 $m$ 값의 범위를 보인다.

〈표 1〉 $t_{scn}(n, m, q)$ 값이 $t_{chk}(n, m, q)$ 값보다 큰 $m$ 값의 범위

〈Table 1〉 Range of $m$ in which $t_{scn}(n, m, q)$ dominates over $t_{chk}(n, m, q)$

| | $m$ 값의 범위 | | | |
|---|---|---|---|---|
| $(\sigma, q)$ | $k=1$ | $k=2$ | $k=4$ | $k=8$ |
| (2,2) | $m \le 6$ | $m \le 5$ | $m \le 3$ | $m \le 3$ |
| (2,3) | $m \le 50$ | $m \le 35$ | $m \le 26$ | $m \le 18$ |
| (2,4) | $m \le 727$ | $m \le 515$ | $m \le 365$ | $m \le 259$ |
| (3,2) | $m \le 15$ | $m \le 11$ | $m \le 8$ | $m \le 6$ |
| (3,3) | $m \le 345$ | $m \le 245$ | $m \le 173$ | $m \le 123$ |
| (4,2) | $m \le 28$ | $m \le 20$ | $m \le 14$ | $m \le 10$ |
| (8,2) | $m \le 120$ | $m \le 85$ | $m \le 60$ | $m \le 43$ |
| (16,2) | $m \le 496$ | $m \le 351$ | $m \le 248$ | $m \le 176$ |
| (32,2) | $m \le 2016$ | $m \le 1426$ | $m \le 1008$ | $m \le 713$ |

알고리즘 2의 경우 $t_{scn}$은 알고리즘 1과 같다. $t_{chk}$는 각각의 패턴 $P_i$에 대한 텍스트 점검 단계에서 검사되는 텍스트 문자수의 기대값 $t^i_{chk}$들의 합으로 주어지며, $t^i_{chk}$는 다음과 같이 주어진다.

$$t^i_{chk}(n_1, n_2, m^{i_1}, m^{i_2}, q_1, q_2) = \frac{(m^i_1-q_1+1)(m^i_2-q_2+1)}{\sigma^{q_1 q_2}} \frac{\sigma}{\sigma-1}(1-\sigma^{-q_1 q_2}) \left\lceil \frac{n_1-m_1+1}{m_1-q_1+1} \right\rceil \left\lceil \frac{n_2-m_2+1}{m_2-q_2+1} \right\rceil$$

간략한 식을 위해 정사각형 가정을 적용하고 패턴의 크기도 모두 같다고 하면 검사되는 텍스트 문자수에 대한 기대값은 다음과 같이 주어진다.

$$t(n, m, q) = t_{scn}(n, m, q) + t_{chk}(n, m, q) = \left(q^2 + k\frac{(m-q+1)^2(\sigma^{q^2}-1)}{(\sigma-1)\sigma^{2q^2-1}}\right) \left\lceil \frac{n-m+1}{m-q+1} \right\rceil^2$$

<표 1>은 몇몇 $k, \sigma, q$ 값들에 대해 $t_{scn}(n, m, q)$ 값이 $t_{chk}(n, m, q)$ 값보다 큰 $m$ 값의 범위를 보이는데, $\sigma$와 $m$ 값이 아주 작을 경우를 제외한 대부분의 경우 여전히 $t_{scn}(n, m, q)$ 값이 $t_{chk}(n, m, q)$ 값보다 크다는 것을 알 수 있다.

앞에서 설명된 내용들을 정리로 요약하면 다음과 같다.

**[정리 2]** 크기가 모두 $m\times m$인 $k$개의 2차원 패턴을 갖는 사전 $D=\{P_1,\ldots,P_k\}$과 크기가 모두 $n\times n$인 2차원 텍스트가 주어지고, 패턴들과 텍스트는 모두 크기가 $\sigma$인 알파벳 $\Sigma$에서 임의로 생성되었으며, $q$는 $m$을 초과하지 않는 정수로 정해질 때, 다음과 같은 사실이 성립한다.

(1) 알고리즘 2에서 검사되는 텍스트 문자수의 기대값 $t(n,m,q)$은 $t_{scn}(n,m,q)$과 $t_{chk}(n,m,q)$의 합으로 주어진다.

(2) $t_{scn}(n,m,q)=q^2\left\lceil\frac{n-m+1}{m-q+1}\right\rceil^2$는 텍스트 탐색 단계에서 검사되는 문자의 수이다.

(3) $t_{chk}(n,m,q)=k\frac{(m-q+1)^2(q^{q^2}-1)}{(\sigma-1)\sigma^{2q^2-1}}\left\lceil\frac{n-m+1}{m-q+1}\right\rceil^2$는 텍스트 점검 단계에서 검사되는 문자의 수의 기대값이다.

(4) 알고리즘 1은 알고리즘 2에서 $k=1$인 특별한 경우이다.

### 3.2 해쉬테이블을 사용한 구현 방법과 기대 수행 시간 분석

텍스트 점검 단계는 단순히 문자들을 차례로 비교하는 방법으로 충분하여 평균 수행 시간은 $\theta(t_{chk})$이 된다. 반면에 *Occurrence*$(s,D)$ 함수는 단순한 구현으로는 수행 시간이 $\theta(|s|\cdot|D|)(|D|=|P_1|+\cdots+|P_k|$는 사전의 크기) 만큼 길어지게 되며, 전체 텍스트 탐색 단계의 수행 시간이 $\theta(t_{scn}\cdot|D|)$가 된다. 여기서 제시되는 구현 방법은 실질적인 텍스트 탐색 단계의 수행 시간을 $\theta(t_{scn})$으로 줄일 수 있는 방법이다.

큰 자연수 $M$이 주어져 있다고 하자 ($M$은 해쉬테이블의 크기보다 큰 임의의 상수). 함수 $f$는 알파벳 $\Sigma$에서 임의로 생성된 $q_1\times q_2$ 사각형에서 집합 $\{0,1,\ldots,M-1\}$으로 사상하는 함수로 다음과 같이 정의한다.

$$f(s)=(\Sigma_{i_1=1}^{q_1}\Sigma_{i_2=1}^{q_2} ord(s[i_1,i_2])*\sigma^{q_1+q_2-i_1-i_2}) \bmod M$$

그리고, 각각의 패턴에 들어 있는 $q_1\times q_2$ 크기의 모든 부분사각형에 대한 $f()$ 함수 값과 인덱스 값들의 집합은 $\tau=\{(f(s),i_1,i_2,i)|s=\rho(P_i,i_1,i_2,q_1,q_2)\wedge 1\le i\le k\wedge 1\le i_1\le m_1^i-q_1+1\wedge 1\le i_2\le m_2^i-q_2+1\}$와 같이 정해진다. 위에서 $ord(x)$ 함수는 알파벳 $\Sigma$에서 정수 집합 $\{0,\ldots,\sigma-1\}$으로 사상하는 임의의 일대일 함수이다. $\tau$는 $|\tau|$개의 리스트를 갖는 해쉬테이블로 구성하며, $f()$ 함수 값을 리스트를 선택하는 키로 사용한다. 즉, 동일한 $f()$ 함수 값을 갖는 항목들은 같은 리스트로 보내진다. *Occurrence*$(s,D)$ 함수는 먼저 $f(s)$ 값을 계산하여 해쉬테이블에서 해당 리스트를 찾은 다음 $s$와 $\rho(P_i,i_1,i_2,q_1,q_2)$에 대한 직접 문자 비교를 통해 $s$가 나타나는 정확한 패턴 상의 위치들을 계산한다.

$f()$ 함수는 기본적으로 Zhu와 Takaoka가 2차원 패턴 정합 알고리즘에서 사용한 해쉬 함수이며, [16]에서 그들은 한 사각형 안에 들어 있는 모든 부분사각형들의 해쉬값이 부분사각형의 크기와 무관하게 전체 사각형의 크기에 선형 비례하는 시간에 계산할 수 있음을 보였다. 그리고, 리스트를 사용하는 해쉬테이블에서 리스트의 수가 실제 자료 항목의 수보다 적지 않을 경우 평균 탐색 수가 2.5를 넘지 않음이 밝혀져 있다 [15]. 위의 두 가지 사실을 종합하면, 패턴 사전 $D$를 위한 해쉬테이블은 평균적으로 $\theta(|D|)$ 시간 안에 계산할 수 있음을 알 수 있다.

*Occurrence*$(s,D)$ 함수의 수행 시간은 함수 $f(s)$의 계산 시간, 해쉬테이블 탐색 시간, 패턴과 텍스트 문자 비교 시간의 세 부분으로 나눌 수 있다. $f(s)$는 $\theta(|s|)$ 시간에 계산될 수 있으며, 해쉬테이블 탐색은 평균적으로 $\theta(1)$ 시간이 걸린다. $s$와 동일한 $f()$ 함수값을 갖는 해쉬테이블 항목의 개수에 대한 기대값은 $|\tau|/M$이 되며, 따라서 문자 비교에 걸리는 평균 시간은 $\theta(|s|\cdot|\tau|/M)$을 넘지 않는다. 앞의 시간들을 더하면 *Occurrence*$(s,D)$ 함수의 수행 시간의 기대값으로 $\theta(|s|+2.5+|s|\cdot|\tau|/M)$을 얻게 되며, 이는 $M\ge|\tau|$인 경우 $\theta(|s|)$가 된다. 따라서, 전체 텍스트 탐색 단계의 기대 수행 시간이 $\theta(t_{scn})$이 된다.

이상의 결과를 요약하면 아래의 정리와 같다.

**[정리 3]** 크기가 모두 $m\times m$인 $k$개의 2차원 패턴을 갖는 사전 $D=\{P_1,\ldots,P_k\}$과 크기가 모두 $n\times n$인 2

차원 텍스트가 주어지고, 패턴들과 텍스트는 모두 크기가 $\sigma$인 알파벳 $\Sigma$에서 임의로 생성되었으며, $q$는 $m$을 초과하지 않는 정수로 정해질 때, 알고리즘 2는 다음과 같은 성능의 기대값을 나타내도록 구현될 수 있다.

(1) 전처리 작업은 $\theta(|D|)$ 시간이 걸린다.
(2) 텍스트 탐색 단계는 $\theta(t_{scn})$ 시간이 걸린다.
(3) 텍스트 점검 단계는 $\theta(t_{chk})$ 시간이 걸린다.
(4) 알고리즘 전체의 수행 시간은 $\theta(|D|+t_{scn}+t_{chk})$ 이다.

알고리즘 2가 추가로 사용하는 메모리는 해쉬테이블을 저장하기 위한 $\theta(|D|)$ 만큼이다.

정리 3의 기대 수행 시간에서 가장 큰 비중을 차지하는 것은 대체로 $t_{scn}$ 항이며, 그 값은 $q_1q_2\left\lceil\frac{n_1-m_1+1}{m_1-q_1+1}\right\rceil\left\lceil\frac{n_2-m_2+1}{m_2-q_2+1}\right\rceil=\theta(q_1q_2n_1n_2/m_1m_2)$이 된다. $\sigma$ 값이 아주 작을 경우를 제외하고는 $q$ 값들을 작은 상수(2 정도)로 정할 수 있으며, 이 경우 전체 수행 시간의 기대값은 $\theta(n_1n_2/m_1m_2)$로 나타낼 수 있다.

이상의 결과들은 텍스트나 패턴의 모양과는 무관하다. 그러나, 사전에 들어 있는 패턴의 크기가 다양하면 알고리즘의 성능은 나빠지기 쉬운데, 그 이유는 수행 시간을 나타내는 식의 분모에 들어 있는 $m_1$과 $m_2$가 각각 패턴의 세로 크기들과 가로 크기들의 최소 값으로 정해지기 때문이다.

### 3.3 실험적 성능 분석

해쉬테이블을 사용하여 구현된 알고리즘 1의 수행 시간에 대한 실험 결과는 <표 2>와 같으며, 비교를 위하여 효율적인 수행 시간을 갖는 알고리즘으로 제안된 Zhu-Takaoka 패턴 정합 알고리즘과 Baeza-Yates와 Régnier의 알고리즘의 수행 시간 결과를 함께 조사하였다. 실험은 솔라리스(Solaris)를 탑재한 펜티움 II PC에서 이루어졌다. 사용된 텍스트의 크기는 1000×1000이며, 사용된 알파벳의 크기 $\sigma$는 흑백 화면의 화소의 종류를 나타내는 2로 고정하였다. $q$의 값은 $m$이 6일 경우는 3, 그 이외의 경우는 4로 고정하였다. 표의 실험 값들은 임의로 생성된 텍스트와 패턴들에 대한 10번씩의 수행 결과에 대한 평균값들이다.

조사된 패턴 크기의 전 범위에서 이 논문에서 제안된 패턴 정합 알고리즘이 지금까지의 가장 빠른 수행 시간을 보여온 Baeza-Yates와 Régnier 알고리즘보다 빠른 수행 시간을 보이며, 이는 패턴의 크기가 증가할수록 더욱 크게 나타난다. 그 이유는 패턴 탐색 단계에서 사용되는 부분사각형의 크기와 패턴의 크기 비율의 차이가 클수록 패턴 탐색 시간을 줄일 수 있으며, 부분사각형의 크기가 패턴과 함께 증가되지 않더라도 우연히 패턴의 한 부분과 일치할 확률이 크지 않기 때문이다. 즉, 대부분의 실험 범위에서 사용되는 부분사각형의 크기가 4×4이며, 여기에는 16개의 이진 기호가 포함되어 우연히 같은 크기의 사각형과 일치할 확률이 $2^{-16}$에 불과하여, 20×20 크기의 패턴이라 하더라도 우연히 텍스트의 4×4 사각형이 패턴에 포함될 확률이 $400*2^{-16}=0.0061$ 정도에 불과하기 때문이다.

〈표 2〉 패턴 정합 알고리즘들의 실험 결과 (msec 단위)
〈Table 2〉 Experimental results of pattern matching algorithms (in milliseconds)

| $k$ \ $m$ | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
|---|---|---|---|---|---|---|---|---|
| 1 | 280 | 316 | 160 | 97 | 65 | 46 | 35 | 27 |
| 2 | 282 | 316 | 161 | 97 | 65 | 46 | 35 | 27 |
| 4 | 286 | 316 | 161 | 97 | 65 | 47 | 36 | 28 |
| 8 | 294 | 316 | 161 | 98 | 66 | 48 | 37 | 30 |
| 16 | 310 | 317 | 162 | 99 | 68 | 50 | 40 | 33 |

<표 3>은 알고리즘 2에서 패턴 수 $k$의 증가가 수행 시간에 미치는 영향을 조사하기 위한 실험 결과를 보이고 있으며, $k$ 값을 제외한 나머지 실험 조건은 앞의 경우와 같다. 패턴의 크기가 작은 경우 ($m=6$)와 패턴의 크기가 큰 경우 ($m>15$) $k$의 증가에 따른 수행 시간의 증가가 나타난다. 패턴의 크기가 작은 경우 패턴의 수가 늘어나면, 패턴의 한 부분과 텍스트의 검사된 부분이 우연히 일치할 확률이 커서 패턴 점검 단계에서 소요되는 시간이 큰 영향을 미친다. 그리고, 패턴이 커지고 또 개수가 크게 늘어나면 패턴을 처리하여 해쉬테이블을 만드는 과정에서의 시간이 많이 소요된다. 조사된 전체 범위에서 패턴 수의 증가가 전체 수행 시간에 미치는 영향은 미미하며, 이러한 결과는 사전에 포

함된 패턴의 크기가 모두 동일할 경우 대체로 비슷한 결과가 기대된다. 그러나, 패턴의 크기에 대한 편차가 심할 경우 $q_1 \times q_2$ 값이 가장 작은 패턴에 의해 제약되므로 수행 시간이 크게 늘어날 가능성을 배제할 수 없다.

〈표 3〉 사전 크기의 영향 조사를 위한 실험 결과 (msec 단위)
〈Table 3〉 Experimental results on the effects of the dictionary size (in milliseconds)

| 알고리즘 \ m | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
|---|---|---|---|---|---|---|---|---|
| Zhu-Takaoka | 610 | 612 | 616 | 608 | 608 | 610 | 608 | 610 |
| Baeza-Yates와 Régnier | 404 | 376 | 373 | 370 | 359 | 352 | 342 | 335 |
| 알고리즘 1 | 280 | 316 | 160 | 97 | 65 | 46 | 35 | 27 |

## 4. 결 론

이 논문에서 제시된 2차원 사전 정합을 위한 실용적인 알고리즘이 갖는 특징은 다음과 같다.

- 이 알고리즘은 임의로 생성된 입력 자료의 경우 실질적으로 텍스트의 일부만 검사하게 되며, 사용되는 검사 과정이 단순하여 짧은 수행 시간을 갖는다.
- 전처리 작업은 패턴에 대해서만 필요하며, 그 수행 시간의 기대값은 사전의 크기에 비례한다. 전체 수행 시간에서 전처리 작업 시간은 사전의 크기가 텍스트의 크기에 비교될 만한 정도가 아니면 무시될 수 있다.
- 이 알고리즘은 패턴이나 텍스트의 모양이나 크기에 제약을 두지 않는다.
- 이 알고리즘은 매우 단순하며, 사용되는 자료 구조는 해쉬테이블 뿐이므로 쉽게 구현될 수 있다.

이 알고리즘은 다음과 같은 확장이 용이하다.

- 이 알고리즘은 텍스트에 대한 특별한 전처리나 자료 구조를 사용하지 않으므로, 동일한 패턴 사전에 대하여 여러 개의 텍스트를 처리할 때 알고리즘의 적용이 단순하다.
- 3차원 이상으로 확장할 경우 $f()$ 함수의 계산, 해쉬테이블 항목의 인덱스 값이 늘어나는 것, 루프의 중첩 정도가 깊어지는 정도의 변화만 있으며, 이들은 쉽게 조정될 수 있다.

## 참 고 문 헌


[1] A. V. Aho and M. J. Corasick, "Efficient string matching : An aid to bibliographic search," *CACM* 18 : 333-340, 1975.

[2] A. Amir, G. Benson, and M. Farach, "An alphabet-independent approach to two-dimensional pattern matching," *SIAM J. on Computing* 23 : 313-323, 1994.

[3] A. Amir and M. Farach, "Two-dimensional dictionary matching," *Info. Proc. Lett.* 44(5) : 233-239, 1992.

[4] A. Amir, M. Farach, R. M. Idury, J. A. La-Pourté, and A. A. Schäffer, "Improved Dynamic Dictionary Matching," *Info. and Comp.* 119 : 258-282, 1995.

[5] R. A. Baeza-Yates, "String Searching Algorithms Revisited," *Workshop in Algorithms and Data Structures*, Lecture Notes in Computer Science 382 : 75-96, Springer-Verlag, Berlin, 1989.

[6] R. A. Baeza-Yates and M. Régnier, "Fast Algorithms for Two Dimensional and Multiple Pattern Matching," *Proc. 2nd Scandinavian Workshop in Algorithmic Theory*, Lecture Notes in Computer Science 447 : 332-347, Springer-Verlag, Berlin, 1990.

[7] T. J. Baker, "A technique for extending rapid exact-match string matching to arrays more than one dimension," *SIAM J. on Computing* 7 : 533-541, 1978.

[8] R. S. Bird, "Two dimensional pattern matching," *Info. Proc. Lett.* 6 : 168-170, 1977.

[9] R. Boyer and S. Moore, "A fast string searching algorithm," CACM 20 : 762-772, 1977.

[10] R. N. Horspool, "Practical fast searching in strings," *Software - Practice and Experience* 10 : 501- 506, 1980.

[11] R. M. Idury and A. A. Schäffer, "Multiple Matching of Rectangular Patterns," *Info. and Comp.* 117 : 78-90, 1995.

[12] R. Karp and M. Rabin, "Efficient randomized pattern-matching algorithms," *IBM J. Res. Dev.*

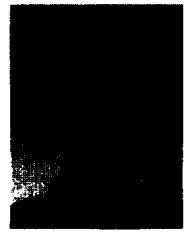
31 : 249-260, 1987.

[13] D. E. Knuth, J. Morris, and V. Pratt, "Fast pattern matching in strings," *SIAM J. on Computing* 6 : 323-350, 1977.

[14] B. Meyer, "Incremental string matching," *Info. Proc. Lett.* 21 : 219-227, 1985.

[15] J. S. Vitter and P. Flajolet, "Average-Case Analysis of Algorithms and Data Structures," in J. van Leeuwen, editor, volume A of *Handbook of Theoretical Computer Science : Algorithms and Complexity*, pp.431-524, Elsevier, Amsterdam, 1990.

[16] R. F. Zhu and T. Takaoka, "A Technique for Two-Dimensional Pattern Matching," *CACM* 32 : 1110-1120, 1989.



**이 광 수**

e-mail : rhee@cs.sookmyung.ac.kr

1981년 서울대학교 계산통계학과 졸업(학사)

1986년 Washington University 대학원 컴퓨터과학과(이학석사)

1990년 Washington University 대학원 컴퓨터과학과(이학박사)

1990년~현재 숙명여대 전산학과 부교수

관심분야 : 알고리즘, 네트워크 보안