근사 주기를 이용한 새로운 랜덤성 테스트 기법

(New Randomness Testing Methods using Approximate Periods)

임 지 혁 † 이 선 호 ††
(Ji Hyuk Lim) (Sun Ho Lee)

김 동 규 †††
(Dong Kyue Kim)

요 약 기존의 패턴 매칭을 이용한 랜덤성 테스트를 개선하기 위하여, 근사 주기에 기반한 새로운 랜덤성 테스트를 제안한다. 근사 주기를 활용하면 랜덤수열에서 비슷한 부분이 반복되는 것을 찾아낼 수 있지만, 계산 시간이 오래 걸리는 단점이 있다. 본 논문에서는 근사주기를 계산하는 시간복잡도를 $O(n^3)$에서 $O(n^2)$으로 줄임으로써, $O(n^2)$의 시간복잡도를 가지는 새로운 랜덤성 테스트를 제안한다. 그리고 AES 암호알고리즘을 이용한 의사 랜덤수열(pseudo random number)과 실제 랜덤수열(true random number)에 제안한 테스트를 적용하여 실험하였다.

키워드 : 랜덤성 테스트, 근사 주기

Abstract In this paper, we propose new randomness testing methods based on approximate periods in order to improve the previous randomness testing method using exact pattern matching. Finding approximate periods of random sequences enables us to search similarly repeated parts, but it has disadvantages since it takes long time. In this paper we propose randomness testing methods whose time complexity is $O(n^2)$ by reducing the time complexity of computing approximate periods from $O(n^3)$ to $O(n^2)$. Moreover, we perform some experiments to compare pseudo random number generated by AES cryptographic algorithms and true random number.

Key words : Randomness test, Approximate period

· 논문은 2008년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업 연구임(KRF-2008-313-D00838)
· 이 논문은 제36회 추계학술발표회에서 '근사 주기를 이용한 새로운 랜덤성 테스트 기법'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 한양대학교 전자통신컴퓨터공학
jhlim@esslab.hanyang.ac.kr
†† 정 회 원 : 한양대학교 전자통신컴퓨터공학
sunholeee@gmail.com
††† 종신회원 : 한양대학교 전자통신컴퓨터공학 교수
dqkim@hanyang.ac.kr

논문접수 : 2009년 12월 24일
심사완료 : 2010년 3월 28일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제6호(2010.6)

1. 서 론

랜덤수(Random number)는 여러 응용 분야가 있지만, 특히 암호 시스템에서 안전한 키를 생성하고 관리하는데 필수적이다. 암호 알고리즘에서 키의 안전성과 밀접한 관련이 있는 중요한 요소이다. 그래서 랜덤수 생성하는 여러 알고리즘들이 제안되었다. 이렇게 인위적인 랜덤수를 의사 랜덤수(Pseudo random number)라고 한다. 이런 의사 랜덤수를 생성하는 알고리즘의 랜덤성(Randomness)을 검증하기 위한 테스트 표준들이 제안되었다. NIST(National Institute of Standards and Technology)에서 제안한 FIPS 140-1[1], FIPS 140-2[2], SP 800-22[3]이 있다. 그 중에서 주기성의 판별을 위하여 정확한 패턴 매칭(Exact Pattern Matching)에 기반한 랜덤성 테스트를 이용하는데, SP 800-22의 Periodic Templates, Aperiodic Templates 테스트는 이를 채택하였다. 정확한 패턴 매칭 대신 에러를 허용하는 근사 패턴 매칭(Approximate Pattern Matching)을 활용하면 랜덤성 테스트의 주기성의 판단의 정확도를 높일 수 있다[4].

본 논문은 근사 패턴 매칭을 이용한 새로운 랜덤성 테스트 방법을 제안한다. DNA 서열의 비슷한 부분 찾는 알고리즘 [5]를 사용하여 기존 근사 주기를 찾는 시간 복잡도를 $O(n^3)$에서 $O(n^2)$으로 단축할 수 있다.

의사 랜덤수열과(Pseudo random number) 실제 랜덤수열(True random number)들에 새로운 랜덤성 테스트를 적용하여 실험한다. 실험 데이터는 의사 랜덤수열인 AES 랜덤수열[6], 실제 랜덤수열인 대기 소음에 의한 랜덤수열[7]으로 사용하였다.

2. 표준화된 의사 랜덤성 테스트

기존에 FIPS 140-1, FIPS 140-2, SP 800-22 등 많은 테스트 방법이 있다. 각각 세부적인 테스트 방법들에 대한 내용은 표 1에 제시하였다. 대부분 방법들이 통계적으로 접근하여 랜덤성 테스트를 하는데, 몇 가지 방법이 본 논문과 유사한 방식인 패턴 매칭으로 랜덤성을 판단 여부를 가린다.

2.1 기존의 정확한 패턴 매칭을 이용한 랜덤성 테스트

랜덤성 테스트 중에서 정확한 패턴 매칭을 이용하여 테스트 하는 방법이 있다. SP 800-22 에서 Periodic Template Matching와 Aperiodic Template Matching 이 패턴 매칭을 사용한다.

2.1.1 Aperiodic Template Matching

SP 800-22[3]에서 제시된 Overlapping Template Matching 방법은 미리 정의된 패턴이 전체 문자열에 얼마나 많이 나오느냐에 따라 랜덤성 여부를 결정한다. 길이 m의 미리 정의된 패턴 x가 있고 길이 m인 window를 이용해서 그 x를 찾는다. 그 window는 전체 문자열 처음부터 시작하여 1만큼 시프트하며 x와 같은 window를 찾아 나간다. 그 뒤 통계학적 처리에 따라서 랜덤성과 비랜덤성(Non-randomness)을 결정한다.

2.1.2 Periodic Template Matching

SP 800-22에서 제시된 Non-overlapping Template Matching 방법은 2.1.1 방법과 비슷하다. 차이점은 window가 전체 문자열의 처음부터 시작하여 x를 찾아 나가는데 2가지 방법으로 진행된다. 만약 찾으면 window는 m만큼 시프트하여 계속 진행되고 못 찾으면 1만큼 시프트하여 계속 진행된다. 그 뒤 통계학적 처리에 따라서 랜덤성과 비랜덤성을 결정한다.

2.1.1, 2.1.2의 제안된 방법으로 실제 랜덤성를 판별하기엔 부족함이 있다. 미리 정의된 패턴들이 한계가 있고, 패턴이 조금씩 변형되어 반복되는 경우 정확한 패턴 매칭으로 찾을 수가 없기 때문에 근사 패턴 매칭을 이용하여 해결해야 한다.

표 1 기존의 랜덤성 테스트 방법

FIPS 140-1, FIPS 140-2	The Monobit Test
	The Poker Test
	The Run Test
	The Long Run Test
SP 800-22	Frequency
	Frequency in block
	The Run Test
	Longest Run of 1s in block
	Binary Matrix Rank Test
	Discrete Fourier Transform Test
	Non-overlapping Template Matching Test
	Overlapping Template Matching Test
	Universal Statistical Test
	Linear Complexity Test
	Serial Test
	Approximate Entropy
	Cumulative Sums Test
	Random Excursions Test
	Random Excursions Variant Test

2.2 근사주기 문제

근사주기(Approximate periods)를 찾는 문제는 몇 가지 있다. 그 중 랜덤성 테스트와 밀접한 문제만을 기술한다.

2.2.1 근사주기를 substring에서 찾는 문제

문자열 x가 주어졌을 때, 최소 distance을 갖는 x의 근사주기 x의 substring p를 찾는다. 이 문제는 편집거리(Edit Distance)를 이용하면 시간복잡도 $O(n^4)$로 해결 할 수 있다. 하지만 시간 복잡도가 크다. 여기서 편집 거리 대신 Hamming Distance를 사용하면 $O(n^3)$으로 줄일 수 있다.

2.2.2 근사주기 문제를 랜덤성 테스트로의 적용

랜덤수열에서 일정하게 반복되는 주기가 나타난다면 그 랜덤수열은 의사랜덤수열일 가능성이 높다. 그 주기를 찾는 방법은 2.2.1에서 언급한 방법으로 구할 수 있다. 만약 어떤 랜덤수열에서 그 안에 존재하는 주기로 이루어진 수열과 랜덤 수열의 편집거리가 상당히 좋게 나왔다면 그 수열에는 근사주기가 존재 하고 그 근사주기의 존재로 인해 그 수열은 의사랜덤수열이라고 말할 수 있다. 그 수열의 주기에 대한 편집거리만 구하게 되더라도 랜덤성 테스트가 가능하다.

2.2.3 근사주기를 찾는 알고리즘의 문제점

근사주기를 이용하여 랜덤성 테스트하는 방법과 SP 800-22과 비교 분석하기 위해서는 적어도 길이는 10^6 랜덤수열을 1000번 테스트를 해야 한다. 하지만 2.2.1의 알고리즘의 시간복잡도로 같은 랜덤수열을 테스트 해보았을 때 많은 시간이 소요되므로 시간복잡도가 향상된 알고리즘이 필요하다.

3. 새로운 근사 주기를 이용한 랜덤성 테스트

본 논문은 랜덤수열에서 비슷한 부분이 반복되는 경우의 랜덤성의 테스트를 위해, 근사 주기를 이용한 새로운 랜덤성 테스트를 제안한다. 근사 주기를 찾으면서 시간복잡도도 낮추기 위해 DNA 문자열에서 비슷한 부분을 찾는 알고리즘을 도입하였다. 2.2.1에서 언급한 알고리즘의 $O(n^3)$ 비해 이 알고리즘의 시간복잡도는 $O(n^2)$으로 시간복잡도를 줄였다.

3.1 새로운 근사 주기를 이용한 테스트 방법

본 논문에서 채택한 근사 주기를 찾는 알고리즘은 동적 프로그래밍 기법을 사용한다. 동적 프로그래밍 기법을 이용한 문제 중에서 편집 거리를 구하는 알고리즘과 비슷하다. 하지만 서로 다른 문자열의 편집거리를 구하는 기존 문제와 달리 같은 문자열에 대해 편집거리를 구하는데 한 문자열을 시프트하면서 최소의 비용을 찾아 나간다. 그 시프트한 거리는 곧 근사주기의 길이가 되며 마지막에 계산된 값은 최종 비용이 된다.

이 근사주기와 최종 비용을 구하기 위해 동적 프로그래밍의 점화식을 사용한다. 그 점화식을 사용하기 위해 몇 가지 용어를 정의한다.

x : 패턴을 찾기 위한 전체 문자열
d[i][j] : x[i]까지 문자열과 x[j]까지 문자열 사이에서 최소 비용
c : 근사주기를 구하기 위해 한 번 시프트하는 비용
Insert cost : Insert하는 비용
Delete cost : Delete하는 비용
Change cost : Mismatch의 경우 Change하는 비용

동적 프로그래밍의 점화식은 다음과 같다

d[i][j+1]=Min(d[i][j+1], d[i][j]+a)

$$a=\begin{cases} c & if\ i=0 \\ \text{Insert cost} & if\ 0<i<j \\ +\infty & otherwise \end{cases}$$

d[i+1][j]=Min(d[i+1][j], d[i][j]+a)

$$a=\begin{cases} c & if\ j=0 \\ \text{Delete cost} & if\ 0<i<j-1 \\ +\infty & otherwise \end{cases}$$

d[i+1][j+1]=Min(d[i+1][j+1], d[i][j]+a)

$$a=\begin{cases} \text{Change cost} & if\ j<j \\ +\infty & otherwise \end{cases}$$

위의 수식을 보면 전형적인 편집거리를 구하는 점화식과 비슷하다. 맨 위에서부터 순서대로 insert, delete, change하는 점화식이다. 하지만 편집거리 문제와 차이점이 있다. 그것은 a값이 고정된 것이 아니라 조건에 따라 변한다. 그 조건은 insert, delete에는 2개, match에는 1개가 존재한다. insert와 delete에서만 있는 i = 0, j = n의 조건은 시프트 할 때의 비용이다. 또 i < j라는 조건이 계속 나오는데 그 조건은 시프트한 결과에 대해서만 계산하게 해준다. 결국 2개의 같은 문자열에 대해 시프트한 문자열과 시프트하지 않은 문자열의 최소 편집거리, 즉 최소 비용을 구하게 되는 것이며 그 때 시프트 거리는 최소 비용을 구할 수 있는 근사주기 길이가 된다.

그림 1은 문자열 AGAGA에 대해 위의 알고리즘을 실행시킨 결과이다.

굵은 선이 최소 비용일 때의 경우를 표시한다. 시프트를 2만큼 할 때가 최소 비용이다. 근사 주기는 AG가 된다. 최소 비용의 구체적인 값은 2차원 배열 d의 값을 보면 알 수 있다. 그림 2은 배열 d에 대한 값들이며 최소 비용을 기준으로 역추적 하여 해당 값에 동그라미를 그렸다. 여기서 insert, delete, change cost는 10으로 시프트 cost는 2로 match cost는 0을 넣어 주었다.

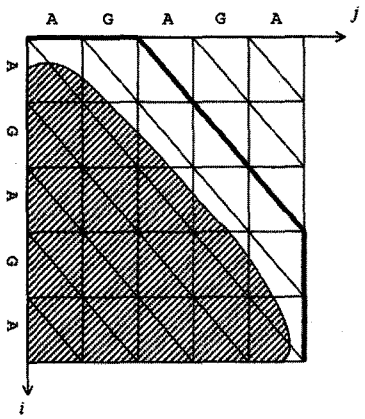

그림 1 문자열 AGAGA에 대한 알고리즘의 동작

②	②	2	2	2	
+∞	12	④	16	8	2
+∞	+∞	14	④	14	2
+∞	+∞	+∞	14	④	2
+∞	+∞	+∞	+∞	14	②
+∞	+∞	+∞	+∞	+∞	②

그림 2 배열 d의 계산된 값

기존의 편집거리 문제처럼 마지막 배열 값이 최소 비용이 아니다. 마지막 시프트까지 계산을 해야 하는데. 마지막 열의 값과 남은 시프트 거리를 더한 것이 최종 최소 비용이다. 그림 2의 맨 위에서부터 계산해보면 (8 +2*4), (14 + 2*3), (4 + 2*2), (14 + 1*2)이다. 가장 작은 값이 (4 + 2*2)인 8이다. 그리고 그 값으로부터 해당 값이 결정된 경로를 역추적하여 표시를 해놓으면 시프트 길이 까지 구할 수 있다.

3.2 새로운 랜덤성 테스트의 척도

3.1의 알고리즘을 사용하면 나오는 결과는 최종 비용과 근사 주기 길이이다. 따라서 랜덤성 테스트의 척도로 최종 거리와 근사 주기 거리가 될 수 있다.

3.1의 알고리즘에서 match가 되었을 때 비용을 0이고 아닌 경우는 어떤 비용 값을 갖게 되는데 이 알고리즘은 최소비용을 구하므로 match가 되려는 쪽으로 나아갈 것이다. 다시 말하면 시프트 하다가도 match 되는게 있으면 시프트를 그만두고 match를 선택하기 때문에 시프트

의 길이, 즉 근사 주기 길이는 별 의미가 없다. 예를 들어 2진수 수열을 3.1의 알고리즘으로 랜덤성을 테스트했을 때 근사 주기 길이는 2이상 넘어가기 힘들다. 왜냐하면 50%의 확률로 match가 되기 때문이다. 그러므로 근사 주기의 길이는 랜덤성 테스트의 척도로는 부족함이 많다. 따라서 근사 주기 길이는 의미가 없다.

그러므로 랜덤성 테스트의 척도로는 근사주기 길이와 상관없는 최종 비용을 이용한다. 같은 문자열으로 실험을 하더라도 insert, delete, change cost가 다르면 최종 비용이 달라지므로 최종 비용에서 위의 3가지 cost들의 평균으로 나눈다. 즉, 불일치가 되는 정도이며 이를 랜덤성 테스트의 척도로 정하고 error라고 정의한다.

$$\text{Error} = \frac{\text{최종비용}}{\text{average(insert, delete, change)}}$$

4. 실험 결과

기존의 랜덤성 테스트와 제안한 랜덤성 테스트를 의사 랜덤수열과 실제 랜덤수열에 적용한 실험 결과를 제시한다.

실험 환경은 Intel® Core™2 CPU 6600 2.40GHz, 2.40GHz, 2.50GB RAM의 사양을 갖는 컴퓨터에서 테스트를 수행하였다.

정확한 패턴 매칭 테스트와 본 논문이 제안한 테스트를 실험하였다.

표 2는 정확한 패턴 매칭을 의사 랜덤수와 실제 랜덤수에 테스트 한 실험한 내용이다. Over는 overlapping template matching을 의미하고 Non-over는 non-overlapping template matching을 의미한다. Pseudo R.N (AES)는 AES에서 발생된 수열[6]이며 True R.N은 random.org에서 얻은 수열[7]이다.

정확한 패턴 매칭으로 실험을 하였을 때 모두 pass한다. 실제 랜덤수의 경우는 모두 통과하였고 AES의 경우 한 번을 제외하고 모두 통과하였다. 즉 한 번만 AES를 의사 랜덤수라고 구분한 것이다. 하지만 NIST SP 800-22를 보면 not pass한 경우의 수가 3% 이내이면 실제 랜덤수라고 보기 때문에 최종적으로는 AES도 실제 랜덤수라고 결과가 나온다.

표 3은 본 논문이 제안한 근사주기를 이용한 랜덤 테스트를 의사 랜덤수와 실제 랜덤수에 테스트 한 내용이다. 1000, 5000, 10000은 테스트한 수열 길이를 의미하고

표 2 정확한 패턴 매칭 결과(100번 실험)

	Exact matching	
	Over	Non-over
Pseudo R.N. (AES)	99% Pass	99% Pass
True R.N.	100% Pass	100% Pass

표 3 근사 패턴 매칭 후 최종비용(100번 실험)

	Using approximate periods		
	1000	5000	10000
Pseudo R.N. (AES)	458	2339	4746
True R.N.	480	2376	4804

각 테스트에 대한 최종 비용을 적어 놓았다. average (insert, delete, change)가 모두 같으므로 생략하였다.

표 3은 근사 패턴 매칭을 이용한 실험 결과이다. 각각 100번 실험 중 가장 좋은 값들만 비교 하였다. Error 값이 낮으면 그만큼 유사한 근사주기가 존재한다. 곧 실제 랜덤수가 아닐 가능성이 크다. 위의 경우를 보면 AES가 실제 랜덤수보다 error 값이 적다. 곧 AES가 실제 랜덤수 보다 랜덤성이 적으며 실제 랜덤수 보다 의사 랜덤수일 가능성이 높다. 그러므로 위의 알고리즘을 사용하여 얻은 최종비용는 랜덤성을 테스트하는데 유효하다고 생각된다.

AES와 실제 랜덤수의 error차이가 두드러지게 크지 않다. 수열의 길이가 커져도 같은 성향을 띤다. 이런 이유에 대해 분석을 해보면 표 2에서와 같이 AES가 실제 랜덤수와 랜덤성이 비슷할 수 있다.

더 높은 정확도를 위해서는 많은 실험을 통해 더 적절한 매개변수를 찾고 통계처리 방법적용과 실제 랜덤수의 최종 비용의 기준이 필요하겠다.

5. 결 론

본 논문은 기존의 패턴 매칭을 이용한 랜덤성 테스트를 개선하기 위해 근사 주기를 활용한 새로운 랜덤성 테스트를 제안했다. 근사 주기를 활용하면 랜덤수열에서 비슷한 부분이 반복되는 것을 찾아낼 수 있지만, 계산 시간이 오래 걸리는 단점이 있다. 본 논문은 DNA 서열에서 비슷한 부분을 찾는 알고리즘을 도입하여 랜덤성 테스트 계산시간을 $O(n^3)$에서 $O(n^2)$으로 줄였다. 위의 실험을 통해 랜덤성 테스트에 대해 유효함을 보였지만 더 좋은 정확도를 위해 앞으로 통계 처리를 도입하고 실제 랜덤성의 기준 등의 추가 연구의 보완이 필요하겠다.

참 고 문 헌

[1] FIPS PUB 140-1: Security requirements for cryptographic modules, NIST, 1994.
[2] FIPS PUB 140-2: Security requirements for cryptographic modules, NIST, 2001.
[3] SP 800-22: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, NIST, May, 15, 2001.
[4] N. Kim, H.L. Obeyesekara and D.K. Kim,

"Comparisons of True Random and Pseudorandom Number Testing," Conference Proceedings of Multimedia, Information Technology and its Applications (MITA 2008), pp.459-462, 2008.

[5] L. Li, R. Jin, P.-L. Kok and H. Wan, "Pseudo-periodic partitions of biological sequences," *BIO-INFOMATICS*, vol.20, no.3, pp.295-306, 2004.

[6] Testing Program for NIST SP 800-22, StsGui ver. 1.8

[7] True Random Number Service, http://www.random.org