변형 비속어 필터링을 위한 Heuristic Methods

Heuristic Methods for A Coined Profanity Fillter

윤태진
Yoon Taijin
부산대학교 컴퓨터공학과
ytj@pusan.ac.kr

ABSTRACT

우리는 변형 비속어 필터링을 연구하면서 DNA 분석에 사용되는 서열정렬 기법을 응용하여 1대1 비교 필터링 문제를 해결하였다. 그러나 실제 비속어 필터링 시스템의 경우 수천에서 수만개의 금칙어 리스트를 가지고 있게 되며 이것을 그대로 적용할 경우 지나치게 많은 연산시간으로 인해 실시간 채팅시스템에 적용하기에는 문제가 있다. 편집거리의 Metric Space를 만족하는 성질을 이용한 근사 문자열 검색을 통해 어느정도 문제를 해결할 수 있었으나 여전히 수만의 금칙어 리스트를 처리하기에는 지나치게 많은 연산량을 필요로 한다. 우리는 이 문제를 해결하기 위하여 여러가지 Huristic Method 접근을 통하여 실시간 채팅시스템에도 적용가능한 비속어 필터링 시스템을 제안하고자 한다.

KEYWORDS Profanity Filter, Heuristic Methods, Phylogenetic Tree

1 서 론

우리는 일반적으로 필터링이 불가능한 변형 비속어 필터링을 위하여 DNA 분석기법 중 하나인 서열 정렬 기법을 응용하여 단어간의 유사도를 측정하는 방식을 사용하여 변형 비속어 필터링을 수행한다[1]. 이것은 1대1 관계의 문자열간의 유사도를 측정하기에 우수한 방법이나 실제 시스템에서는 수천에서 수만의 금칙어가 적용되게 된다. 단어간 10000번의 서열정렬 값을 측정하기 위해 약 0.5초의 시간이 필요한데 우리 시스템은 약 6천여개의 금칙어를 사용하므로 단어당 약 0.3초의 시간을 필요로 한다. 하나의 문장을 필터링하기 위해 수십번의 단어 검색이 필요하므로 실시간 채팅 시스템에는 도저히 사용할 수가 없다.

우리는 이 문제를 해결하기 위해 편집거리(Edit Distance)의 Metric Space를 만족하는 성질을 이용한 근사 문자열 검색을 통하여 검사가 필요한 단어를 비약적으로 줄이는데 성공하였다 [3]. 표 1는 해당 시스템을 이용하여 비속어 필터링을 수행한 수행 결과이다. 단어당 0.3초에서 0.0102초까지 단축되어 상당한 성능향상을 보인 것을 알 수 있다. 실시간 채팅에 사용되는 하나의 문장을 필터링하기 위해 20번 정도의 단어 검색이 필요하다고 할때 하나의 문장을 필터링하는데 걸리는 시간은 0.2초로 납득할 수 있는 속도라고 할 수 있다. 그러나 이건은 1대 1채팅 상황에서 클라이언트간에 순수한 채팅만 이루어지는 경우이다. 실제 온라인게임에서는 1초에도 수 수십개의 문장이 처리되는 경우가

Copyright © Graphics Application Lab. Pusan National University

있으며 게임을 구동하기 위해 시스템 자원을 대부분 사용하기 때문에 채팅 시스템에 그렇게 많은 자원을 할당할 수는 없다. 게다가 클라이언트 단위에서 필터링이 이루어질 경우 클라이언트를 변조하여 비속어 필터링 시스템을 무력화 시킬 수 도 있기 때문에 서버에서 중앙 집중형으로 처리할 경우 여전히 많은 연산량이 요구된다고 할 수 있다.

표 1. pivot 수에 따른 검색 효율성 실험 - A, C는 이미 최적화가 되어 실험하지 않았고 B의 경우 최적화 영역과 멀어 실험을 수행하지 않았음.

	일반단어 1000		일반단어 60000		비속어 6000	
pivot수	시간	평균결과수	시간	평균결과수	시간	평균결과수
10	0.0169	505	B	B	0.0716	2145
20	0.0110	331	B	B	0.0388	1214
30	0.0084	236	B	B	0.0244	762
40	0.0072	179	B	B	0.0176	541
50	0.0064	141	B	B	0.0152	441
60	0.0058	105	B	B	0.0132	356
70	0.0054	77	B	B	0.0114	287
80	0.0056	70	0.1272	4665	0.0109	262
90	0.0058	56	0.1037	3703	0.0102	211
100	0.0060	48	0.0935	3278	0.0104	182
110	A	A	0.0849	2983	C	C
120	A	A	0.0748	2558	C	C
130	A	A	0.0668	2163	C	C
140	A	A	0.05741	1855	C	C
150	A	A	0.0552	1728	C	C
160	A	A	0.0517	1613	C	C
170	A	A	0.0515	1535	C	C

이 보고서에서는 연산량을 줄이기 위하여 2가지 Huristic Methods를 제안하고자 한다. 사람이 단어를 인지하는 방법을 고려한 단어 필터링 방식과 유사한 단어를 그룹으로 묶어서 검사수를 줄이는 방법을 제안한다.

Copyright © Graphics Application Lab. Pusan National University

2 단어의 인지 방법을 이용한 단어 필터링 기법

비속어 필터링을 피하기 위해 변형 비속어를 입력할때 가장 많이 하는 것은 단어의 사이에 문자를 삽입하는 방법이다. "개자식"을 입력하고 싶다면 "개ㅅ자ㅅ식"을 입력한다던지 하는 방식을 사용하게 된다. 그에 반해 글자를 제거하는 방식은 그다지 사용되지 않는다. "개식", "자식", "개자" 등은 의미를 제대로 전달하기 어렵기 때문이다. 그리고 사람은 단어의 첫자와 끝자를 단어를 인식하는데 가장 중요하게 사용한다. 예를 들어 "안드로이드"라는 단어를 "안로드이드"나 "안드이로드"라고 적어도 단어를 인식하는데 큰 문제를 느끼지 못한다. 즉 첫자와 끝자는 단어에서 가장 중요한 부분으로 이를 이용하여 비속어 필터링에 필요한 서열정렬 측정횟수를 줄일 수 있다.

기존의 금칙어 데이터구조는 하나의 다차원구조 R*tree에 모든 단어를 집어 넣어서 구성되기 때문에 효율성을 높이기에는 한계가 있었다. 그래서 우리는 이 첫자와 끝자의 아이디어를 이용하여 여러 개의 R*tree를 구성하는 방법을 사용하고자 한다. 단어를 여러 개의 그룹으로 나누는 기준은 단순한데 단어의 첫자와 끝자를 추출한 뒤에 문자열을 표준형 변환을 통해서 통일화 시킨다. 이 표준형 변환으로 인해 대부분의 자소 변형을 무력화 시킬 수 있다. 그룹화된 단어군은 이 첫자와 끝자의 표준형을 이용한 Hash를 이용하여 빠르게 접근이 가능하다.

이렇게 단어를 그룹화 하더라도 하나의 그룹에는 많은 단어가 포함될 수 있다. 일정 수 이상의 그룹은 기존의 방법을 이용한 R*tree를 구성하여 같은 방법을 통해 검사가 필요한 후보군을 추출하게 된다. 그러나 단어의 수가 20개 이하일 경우 굳이 R*tree를 구성하지 않고 전수 검사를 수행하는 편이 효율성이 높아지게 된다.

이 필터링 방식은 특히 문장단위의 필터링 방식에 유용하게 사용할 수 있다. 실제 채팅에서 사용자가 띄어쓰기를 정확하게 지키지 않은 경우가 빈번하게 일어난다. 특히 비속어를 사용할 경우 정확한 국어 문법을 지키기를 기대하기에는 어려움이 있다. 이 문제를 해결하기 위하여 우리는 일정 글자수 n을 지정하여 한글자씩 문장을 이동해 가며 비속어를 찾는 방법을 사용한다.

표 2. 문장 필터링시 실제 필터링되는 단어와 횟수 - n = 5

입력문장			야이개자식아		
번호	검사단어	Hash Index	번호	검사단어	Hash Index
1	야이	ㅇㅏㅇㅣ	2	야이개	ㅇㅏㄱㅐ
3	야이개자	ㅇㅏㅈㅏ	4	야이개자식	ㅇㅏㅅㅣㄱ
5	이개	ㅇㅣㄱㅐ	6	이개자	ㅇㅣㅈㅏ
7	이개자식	ㅇㅣㅅㅣㄱ	8	이개자식아	ㅇㅣㅇㅏ
9	개자	ㄱㅐㅈㅏ	10	개자식	ㄱㅐㅅㅣㄱ

Copyright © Graphics Application Lab. Pusan National University

위의 표 2는 "야이개자식아"라는 문장에서 "개자식"이라는 비속어를 필터링하기 위하여 수행되는 필터링 횟수와 입력되는 문자열, 그리고 그 문자열의 Hash Index를 보여주고 있다. "개자식"을 추출할때까지 약 10번의 입력이 이루어졌는데 이 과정에서 각각 입력된 Hash Index의 경우 비속어의 첫자와 끝자로 쓰이는 문자열은 없는 것을 알 수 있다. 즉 개자식에 도달할때까지 아주 적은 연산량을 가지는 Hash연산만 수행되었다는 것이다. 그리고 "개자식"에 도달하여 "개식"을 Hash Index로 사용하여 정확하게 단어를 필터링 할 수 있다. 띄어쓰기가 전혀 이루어지지 않은 문장에서도 빠르게 비속어를 필터링할 수 있는 것이다.

3 변형비속어의 **Phylogentic Tree**

3.1 근사 문자열 검색의 한계

근사문자열 검색을 이용한 변형 비속어 필터링에서 가장 큰 문제점은 검색 반경 설정의 문제이다. 검색 반경을 증가시킬때마다 후보군의 개수가 기하급수적으로 증가하기때문에 3 4이상의 검색 반경을 주는것은 무리가 있다. 그래서 우리는 이러한 문제를 해결하기 위하여 비속어의 phylogenetic tree를 구성하여 입력단어와 유사한 단어를 찾는 실험을 수행하였다.

이 실험은 기존의 방법에 비해 효율이 현격히 떨어졌다. 기존의 시스템에 비해서 검색시간이 4 5배 정도 오래걸렸을 뿐더러 검색의 정확성도 50% 정도로 매우 낮았다. 문제의 원인은 크게 2가지가 있었다. 첫째, 비속어의 싱글톤 문제이다. 상호 유사성이 전혀 없는 비속어가 많기 때문에 유사도를 이용한 tree를 구성하고자 하여도 충분한 유사도를 보이지 않는 단일 단어들이 수많은 리프 노드를 만들어 tree의 구조가 불균형한 형태를 만들게 되었다. 둘째로 비속어 간의 진화 관계성이 전혀 없는 것을 간과한 점이다. 생물은 하나의 근원에서 진화를 시작하였기에 생물종 전체를 진화 tree를 구성할 수 있으나 비속어의 경우 한 비속어에서 파생된 변형 비속어들을 제외하고는 단어간에 하등의 연관성이 존재하지 않는다. 이러한 단어군을 무리하게 하나의 진화 tree로 묶어서 구성하더라도 정상적인 형태의 Tree 구조를 기대하기는 어렵다. 예를 들어 "시발"과 "개새끼"의 공통된 조상을 찾고자 해도 그러한 것은 존재하지 않는다. 물론 "개시발", "시발새끼" 같은 단어를 공통 조상이라고 생각할 수 도 있으나 공통된 조상이라기 보다는 형제에 가깝다고 할 수 있다.

비속어 데이터베이스의 단어들은 수많은 싱글톤들과 몇몇 주로 사용되는 욕설들의 변형 비속어 그룹들로 이루어지게 된다. 싱글톤의 경우 변형비속어가 잘 발견되는 단어가 아니기에 굳이 변형 비속어 검색 방법을 사용하지 않고 일반적인 swear filtering방식을 이용하여 처리하는 방법을 사용할 경우 필터링의 효율성을 높일 수 있을 것이다. 그러나 이 방법은 검색의 효율성을 높일 수는 있으나 해당 욕들의 변형 비속어 사용이 활발해질 경우 대처가 어렵다.

phylogenetic tree를 이용한 변형 비속어 필터링의 경우 비속어 전체를 tree로 묶는 방법은 그다지

Copyright © Graphics Application Lab. Pusan National University

효율적이 못하다는 것을 이미 이야기 하였다. 반면에 변형을 통한 서로간의 진화 관계가 확실한 변형 비속어에 대해서는

3.2 Phylogentic Tree를 이용한 검색 효율성 향상

금칙어 리스트에는 중요한 몇몇 비속어에 대하여 변형 비속어를 추가로 입력하였으며 이를 통해 변형 비속어의 필터링 정확도를 높일 수 있다. 그러나 이들 변형 비속어를 각각 독립된 원소로서 취급하여 필터링을 수행할 경우 매우 비효율적인 필터링이 이루어지게 된다. 해당 그룹의 변형 비속어가 입력될 경우 금칙어의 모든 변형비속어를 다 검사하게 되기 때문이다.

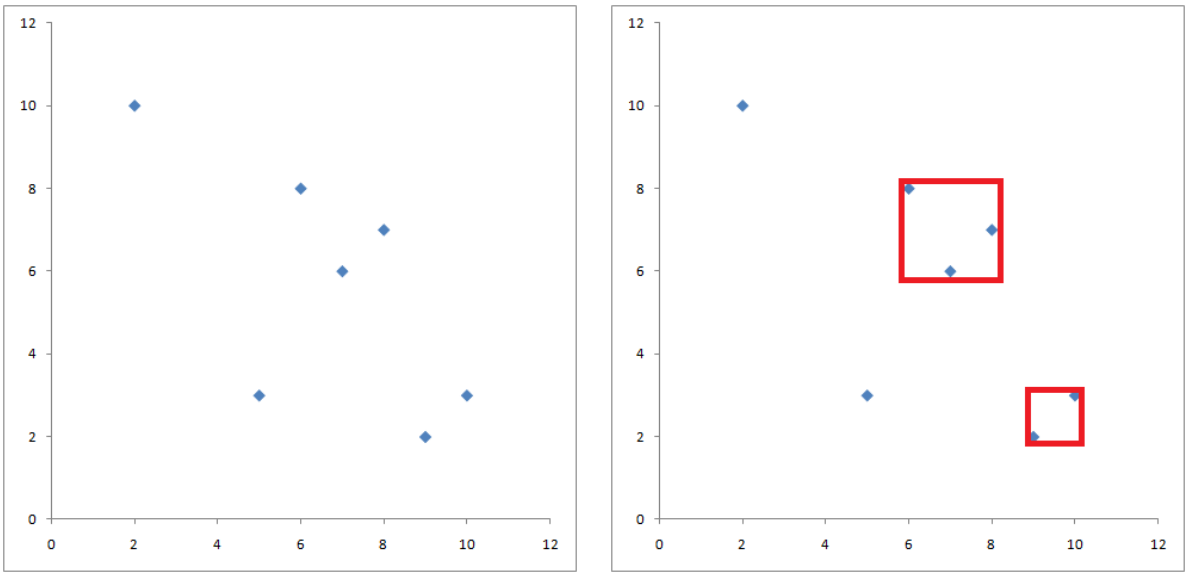

그림 1. 유사한 단어끼리 그룹화하여 하나의 큰개체로 취급하게 된다.

이 문제를 해결하기 위해서는 특정비속어의 변형 비속어를 금칙어에 추가할 경우 각각을 원소로 취급하지 않고 해당 그룹을 하나의 객체로서 취급하는 방법이 있을 수 있다. 원래 비속어와 입력된 변형비속어를 하나의 그룹으로 묶고 R*tree 좌표를 계산할때 pivot과의 편집거리 측정에서 가장 높게 나온 값과 가장 낮게 나온값을 범위로 가지는 객체를 만드는 것이다. 그림 1는 실제 시스템에서 적용되는 2차원 형태의 예를 보여주고 있다.

단어의 그룹을 하나의 객체로 취급함으로써 원소의 갯수를 줄일 수 있었다. 그러나 이 객체를 효과적으로 다루기 위한 방법을 고려할 필요가 있다. 먼저 객체의 검사 방법이다. 단순히 기본단어와 입력단어의 유사도를 측정하는 방법을 사용한다면 굳이 변형비속어를 금칙어에 추가한 의미가 없다. 그렇다고 모든 그룹의 단어와 검사를 수행하는것도 비효율적인 방법이다. 먼저 일반 욕과 입력단어의 검사를 수행하고 유사도 수치가 지나치게 낮을 경우는 검사를 수행할 필요가 없고 유사도 수치가 비속어 기준치를 넘을 경우도 검사할 필요가 없다. 상대유사도가 50% 이상 75%(임계값) 이하의 비속어

Copyright © Graphics Application Lab. Pusan National University

에 한하여 그룹내의 변형 비속어와 검사를 수행할 필요가 있다.

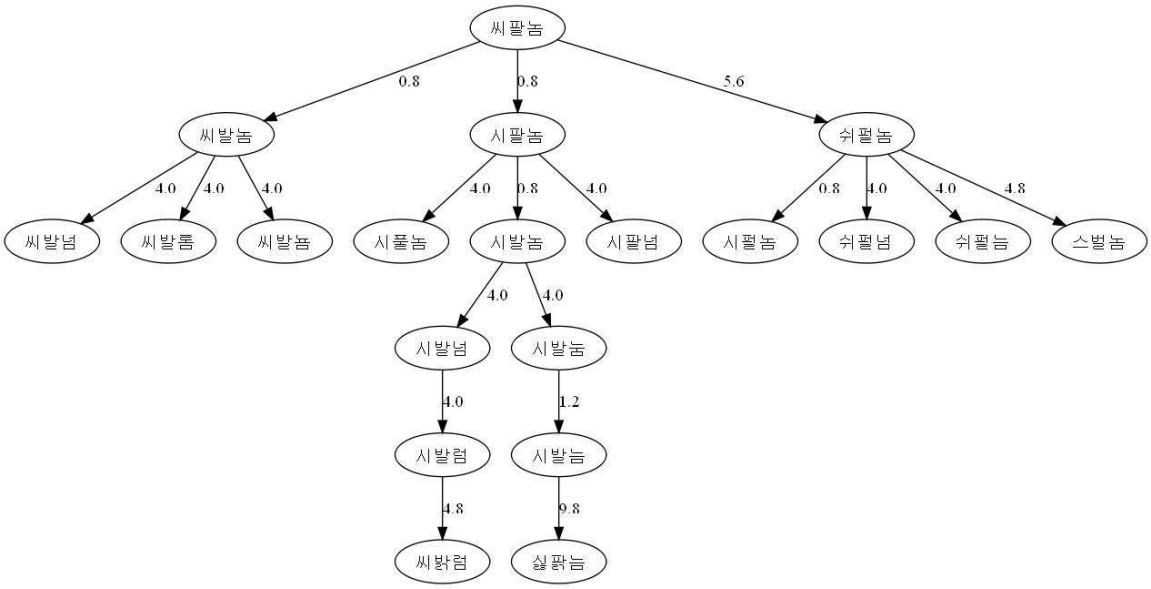

그림 2. 비속어와 그 변형으로 구성한 Phylogenetic Tree

그룹내의 단어를 검사할때도 모든 단어를 검사할 경우 연산시간이 매우 오래걸리게 된다. 이때 이 검사시간을 줄이기 위해서 우리는 phylogenetic tree를 이용한다. 비속어의 변형을 진화로 보고 비속어의 진화 트리를 구성하는 것이다. 이 진화 트리를 traverse하여 가장 유사도가 높은 단어를 찾는 방식을 사용하게 된다. 그림 2는 실제 비속어와 그 변형비속어들을 통하여 구성한 phylogenetic tree의 예이다. tree구조를 이용함으로써 단어의 검사 시간을 매우 단축시킬 수 있다.

4 결론

이 보고서는 변형 비속어 필터링의 속도문제를 해결하기 위한 huristic methods를 제안하였다. 제안된 방법은 다음과 같다.

1. 단어의 인지 방법을 이용한 단어 필터링 기법 : 사람이 단어를 인지하는데 가장 중요하게 보는것은 첫자와 끝자이다. 이것을 이용하여 단어를 첫자와 끝자가 같은 것끼리 그룹화하여 따로 R*tree를 구성하여 단어당 검사시간을 단축하고 문장단위의 검사에서도 효율성을 높일 수 있다.

2. 변형비속어의 Phylogentic Tree : 변형 비속어를 따로 원소로 취급하지 않고 일반 비속어와 변형비속어 들을 하나의 그룹으로 구성하여 객체수를 줄여 검사횟수를 감소시켰다. 더불어 그룹을

Copyright © Graphics Application Lab. Pusan National University

phylogenetic tree의 형태로 만들어 그룹내의 단어를 검사할 때에도 효율성을 증가 시켰다.

시스템의 시간적 효율성을 높이는데는 좋은 방법론들이다. 그러나 그 만큼 필터링의 범용성이 감소하였으므로 경우에 따라서는 필터링을 간단하게 무력화 할 수 있는 방법이 존재할 수도 있다. 추후 연구를 통하여 이 방법들의 취약점을 연구하고 보안하는 연구가 필요할 것이다. 그리고 일반단어를 필터링하는 문제에 대해서도 추가적인 연구가 필요하다[2].

참고 문헌

1. "http://en.wikipedia.org/wiki/swear_filter," .
2. "http://en.wikipedia.org/wiki/scunthorpe_problem," .
3. Gonzalo Navarro and Edgar Chávez, "A metric index for approximate string matching," *Theor. Comput. Sci.*, vol. 352, no. 1, pp. 266–279, 2006.
4. Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger, "The r*-tree: an efficient and robust access method for points and rectangles," in *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 1990, pp. 322–331, ACM.
5. Antonin Guttman, "R-trees: a dynamic index structure for spatial searching," *Readings in database systems*, pp. 599–609, 1988.
6. Shekhar Dhupelia, "Designing a vulgarity filtering system," in *Game Programming Gems 5*. 2005, Charles River Media.
7. Lai C, "An empirical study of three machine learning methods for spam filtering," *Know.-Based Syst*, vol. 20, no. 3, pp. 249–254, 2007.
8. Ramachandran A Feamster N and Vempala S, "Filtering spam with behavioral blacklisting," *In Proceedings of the 14th ACM Conference on Computer and Communications Security (Alexandria, Virginia)*, pp. 342–351, 2001.
9. Kyo hyeon Park and Jee hyong Lee, "Developing a vulgarity filtering system for online games using svm," *In Proceedings of the Korean Institute of Information Scientists and Engineers Autumn 2006*, 2006.
10. Chang-Keon Ryu, Hyong-Jun Kim, Seung-Hyun Ji, Gyun Woo, and Hwan-Gue Cho, "Detecting and tracing plagiarized documents by reconstruction plagiarism-evolution tree," *Computer and Information Technology, 2008. CIT 2008. 8th IEEE International Conference on*, pp. 119–124, July 2008.
11. Hyong-Jun Kim, Chang-Keon Ryu, and Hwan-Gue Cho, "A detecting and tracing algorithm for unauthorized internet-news plagiarism using spatio-temporal document evolution model," in *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, New York, NY, USA, 2009, pp. 863–868, ACM.
12. Chang-Keon Ryu, Hyong-Jun Kim, and Hwan-Gue Cho, "Reconstructing evolution process of documents in spatio-temporal analysis," in *ICCIT '08: Proceedings of the 2008 Third International Conference on Convergence and Hybrid Information Technology*, Washington, DC, USA, 2008, pp. 136–142, IEEE Computer Society.
13. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, and D.J., "Basic local alignment search tool.," *Journal of Molecular Biology.*, vol. 215, 1990.
14. A. Apostolico, "The myriad virtues of subword trees," *Combinatorial Algorithms on Words*, pp. 85–96, 1985.

Copyright © Graphics Application Lab. Pusan National University

15. W. A. Burkhard and R. M. Keller, "Some approaches to best-match file searching," *Commun. ACM*, vol. 16, no. 4, pp. 230–236, 1973.

16. Sreenivas Gollapudi and Rina Panigrahy, "A dictionary for approximate string search and longest prefix search," in *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, New York, NY, USA, 2006, pp. 768–775, ACM.

17. Trinh N. D. Huynh, Wing-Kai Hon, Tak-Wah Lam, and Wing-Kin Sung, "Approximate string matching using compressed suffix arrays," *Theor. Comput. Sci.*, vol. 352, no. 1, pp. 240–249, 2006.

18. Marios Hadjieleftheriou, Nick Koudas, and Divesh Srivastava, "Incremental maintenance of length normalized indexes for approximate string matching," in *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*, New York, NY, USA, 2009, pp. 429–440, ACM.

19. W. A. Burkhard and R. M. Keller, "Some approaches to best-match file searching," *Commun. ACM*, vol. 16, no. 4, pp. 230–236, 1973.

Copyright © Graphics Application Lab. Pusan National University