대어휘 연속음성인식을 위한 서브네트워크 기반의 1-패스 세미다이나믹 네트워크 디코딩*

정민화(서강대), 안동훈(서강대)

<차 례>

1. 서 론
2. 언어 모델 네트워크
3. 서브네트워크 기반의 탐색 네트워크 구성
4. 세미다이나믹 네트워크 디코딩
4.1. 세미다이나믹 네트워크 관리
4.2. 서브네트워크 미리 로드
4.3. 활성화 빈도수 추정
4.4. 서브네트워크 캐싱
4.5. 서브네트워크 프로파일링
5. 테일 공유 알고리즘
5.1. 네트워크 최소화 알고리즘
5.2. 테일 공유 알고리즘
5.3. 언어 모델 네트워크의 널 - 전이 아크 제거
6. 실험 결과
6.1. 실험 환경
6.2. 서브네트워크 기반의 탐색 네트워크 구조 분석
6.3. 디코더 성능
6.4. 서브네트워크 캐싱 성능
6.5. 프로파일링을 통한 서브네트워크 캐싱 성능 향상
7. 결 론

<Abstract>

1-Pass Semi-Dynamic Network Decoding Using a Subnetwork-Based Representation for Large Vocabulary Continuous Speech Recognition

Minhwa Chung, Dong-Hoon Ahn

In this paper, we present a one-pass semi-dynamic network decoding framework that inherits both advantages of fast decoding speed from static network decoders and memory efficiency from dynamic network decoders. Our method is based on the novel language model network representation that is essentially of finite state machine (FSM). The static network derived from the language model network [1][2] is partitioned into smaller subnetworks which are static by nature or self-structured. The whole network is dynamically managed so that those subnetworks required for decoding are cached in memory. The network is near-minimized by applying the tail-sharing algorithm. Our decoder is evaluated on the 25k-word Korean broadcast news transcription task. In case of the search network itself, the network is reduced by 73.4% from the tail-sharing algorithm. Compared with the equivalent static network decoder, the semi-dynamic network decoder has increased at most 6% in decoding time while it can be flexibly adapted to the various memory configurations, giving the minimal usage of 37.6% of the complete network size.

* Keywords: Speech Recognition, Semi-dynamic Network Decoding, Language Model Network, Subnetwork Caching, Tail-Sharing Algorithm

* 이 논문은 과학기술부 특정연구개발 과제인 뇌신경정보학 과제의 지원(M1-0107-01-0003)으로 수행되었습니다.

1. 서 론

스태틱 네트워크를 이용한 탐색 네트워크는 많은 음성 인식 시스템의 초기 단계에서 사용되고 있다. 이는 네트워크의 상태에 관계없이 비터비 디코딩(Viterbi decoding) 알고리즘의 효율적인 구현이 가능하다는 점과 이에 따른 응답 시간의 단축에 기인하고 있다. 그러나 태스크가 복잡하게 되면, 모든 지식원 들을 네트워크에 미리 통합시켜야 한다는 점 때문에 메모리 사용 면에서 문제점을 안고 있다. 이러한 공간적인 제약을 해결하기 위해, [3]에서는 후자 트리(successor tree)와 널 노드(null node)의 개념을 도입하여 바이그램(bigram)을 효과적으로 표현할 수 있는 방법을 제안하였으며, [1][2]에서는 이를 보편화하여 N-그램을 언어 모델 네트워크로 표현하는 방법으로 확장하였다. 그러나 이러한 표현방법이 대규모의 언어 모델을 효과적으로 표현할 수 있는 방법을 제시하고 있다는 점은 바람직하나, 디코딩 측면에서 두 가지 문제점을 찾아볼 수 있다. 한가지는 각 후자 트리의 크기는 크지 않더라도 그 개수는 트라이그램 이상의 N-그램과 보다 많은 수의 어휘가 사용될 때에는 메모리 요구 량이 급증한다는 점이다. [3] 및 [4]에서는 전체 네트워크를 오토마타(automata)와 트랜스듀서(transducer)의 입장에서 최소화할 수 있는 알고리즘을 제안하였지만 이 알고리즘을 적용하기 위해서는 상당한 계산 자원이 필요할 뿐 아니라, 최소화된 네트워크도 본래의 크기에 비해 많이 작긴 하지만 여전히 많은 메모리가 필요하다는 점도 간과할 수 없다. 두 번째 문제점은 이렇게 구성된 인식 네트워크를 모두 메모리에 로드하더라도 프루닝(pruning) 작업을 통해 네트워크의 일부만이 디코딩과정에 사용되기 때문에 전체 네트워크를 메모리에 유지해야 한다는 사실은, 특히 대규모 태스크의 경우에 있어서 매우 비효율적이다.

위의 문제점들을 적극적으로 해결하기 위해서는 디코딩 과정에서도 네트워크를 그 상태에 따라 그 일부를 메모리에 생성하거나 철수시키는 동적인 관리 방법이 필요하다. 그러나 현재의 다이나믹 디코딩 방법[5][6]은 디코딩 알고리즘의 구현에 많은 변화를 가져왔다. 무엇보다도 언어모델 미리참조(LM lookahead)를 동적으로 구성하거나[7], 많은 수의 노드와 아크들을 직접 관리해야 하는 부담, 그리고 네트워크의 각 노드마다 별도의 탐색 가설들을 유지하기 위한 노력(per-state stack management [8]) 등은 디코더의 구성을 매우 복잡하게 만들었다.

이 논문에서는 스태틱 네트워크의 공간적인 비효율적인 사용 측면을 동적인 네트워크 관리 기법으로 해결할 수 있는 방법을 세미다이나믹 네트워크 디코딩 방법을 제안한다. [1]의 언어 모델 네트워크를 기반으로 하여 탐색 네트워크를 구성한 후, 언어 모델 네트워크의 각 언어 모델 히스토리(LM history)를 중심으로 탐색 네트워크를 분할(partition)한다. 여기서 네트워크 분할 단위가 서브네트워크(subnetwork)에 해당하며 전체 네트워크는 이 서브네트워크 단위로 구성된다. 서브네트워크 자체는 생성시 자동으로 스태틱 네트워크로 구성하도록 고안하였으며,

디코딩 시에는 필요한 서브네트워크들을 메모리에 캐싱 하는 동적인 방법을 사용하였다. 한편, 이렇게 구성된 네트워크는 내부적으로 많은 중복된 정보[1]를 포함하고 있다[3][4][9][10]. 여기에서는 상당한 규모의 계산 복잡도를 갖는 WFST 알고리즘들[4][9]을 적용하는 대신, 테일 공유 알고리즘[10]을 개선한 방법을 적용하였다.

이 논문의 구성은 다음과 같다. 2장에서는 [1]에서 소개된 언어 모델 네트워크에 대해 정리하며 3장에서는 서브네트워크 기반의 인식 네트워크를 구성하는 방법에 대해 알아보도록 한다. 4장에서는 서브네트워크 기반의 세미다이나믹 네트워크 디코더를 설계하며, 5장에서는 서브네트워크 기반의 탐색 네트워크의 중복 정보를 효과적으로 제거할 수 있는 개선된 테일 공유 알고리즘을 기술한다. 마지막으로 6장에서 실험 결과를 정리하고, 7장에서 결론을 맺도록 한다.

2. 언어 모델 네트워크

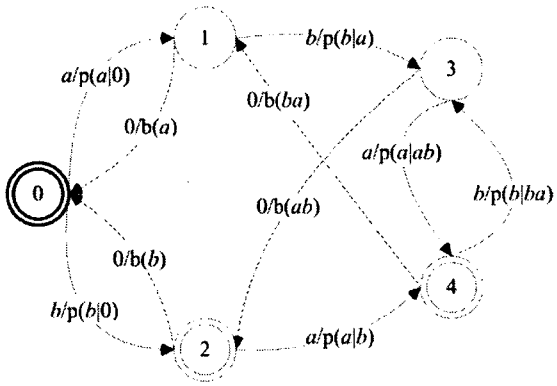
언어 모델 네트워크[1][2]는 임의의 언어 모델을 인식 단계에서 동일하게 처리하기 위해 만들어진 오토마타 기반의 표현 방법이다. 일반적으로 언어 모델 확률 값은 현재 단어와 주어진 언어 모델 히스토리로부터 계산된다. 가령, 단어열 W=w1w2…wn 에 대한 언어 모델 확률 값은 다음과 같이 계산된다.

$$\begin{aligned} p(\mathrm{W}) &= p(w_1 w_2 \cdots w_n) \\ &= p(w_1) \times p(w_2 \mid w_1) \times \cdots \times p(w_n \mid w_1 \cdots w_{n-1}) \\ &= \prod_i p(w_i \mid w_1 \cdots w_{i-1}) = \prod_i p(w_i \mid h_i) \end{aligned}$$

모든 언어 모델은 이러한 두 가지, 입력 단어(w_i) 및 히스토리(h_i)를 기반으로 하고 있다. 한편, 단어와 LM 히스토리의 조합은 또 하나의 LM 히스토리가 되는 재귀적인 구조로 되어 있다. 따라서 언어 모델 네트워크는 단어와 LM 히스토리로 구성된 LM 컨텍스트를 노드로 정의하며 컨텍스트 간의 전이는 아크로, 히스토리에 대한 입력 단어의 조건부 확률 값을 아크의 가중치로 정의한다. 백오프(backoff) 아크를 도입하여 학습에 나타나지 않은 단어열에 대한 확률 값을 제공하도록 하였다. 단어열의 언어 모델 값은 언어 모델 네트워크에서 각 단어에 해당하는 경로의 가중치를 누적하여 계산할 수 있다. <그림 1>은 두 단어 a, b에 대한 백오프

1) 네트워크의 중복 정보(redundancy)는 네트워크의 여러 경로가 동일한 HMM 열(sequence)을 출력하는 경우를 가리킨다. 이러한 중복성은 대부분 언어모델, 음향모델 등의 계층 구조를 가진 지식원 사이의 결합시, 그리고 이들 모델 학습에 사용되는 통계적인 추정 방법의 사용에 따른 필연적인 결과이다.

트라이그램 네트워크를 나타낸다. 점선은 백오프 아크를 나타내며, 굵은 선의 원은 시작 노드, 이중원은 종료 노드를 가리킨다.



<그림 1> 백오프 트라이그램 네트워크

3. 서브네트워크 기반의 탐색 네트워크 구성

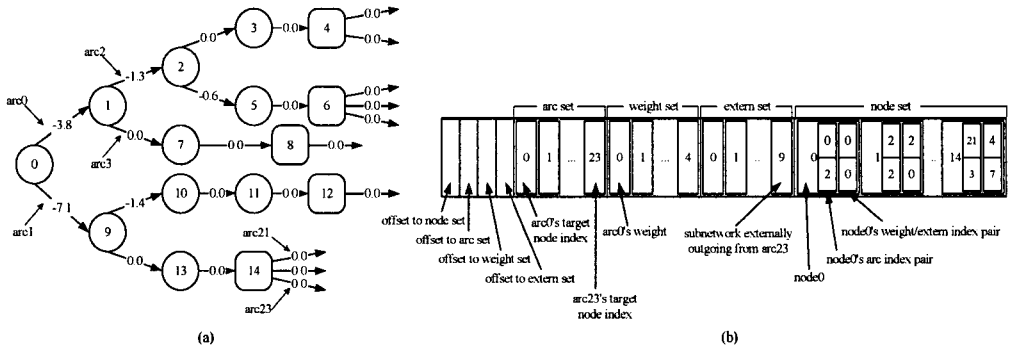
이렇게 구성된 언어 모델 네트워크는 각 컨텍스트 마다 후자 트리(successor tree)를 구성함으로써 전체 탐색 네트워크를 구성하게 된다[1]. 탐색 네트워크는 이제 네트워크의 관리 단위인 서브네트워크단위로 분할되는 데, 필요한 경우 탐색 네트워크의 생성과정과 병행함으로써 네트워크 생성에 필요한 메모리를 더욱 줄일 수 있다. 네트워크 관리비용은 크게 두 가지 요소로 나누어진다. 한가지는 서브네트워크를 메모리에 생성하는 데 드는 비용을 최소화하는 일이며, 다른 한가지는 서브네트워크의 생성 회수를 최소로 유지하는 일이다. 두 번째 문제에 대해서는 4절에서 알아보도록 하고 여기서는 서브네트워크의 생성에 대해 논하도록 한다.

서브네트워크의 생성 비용에 관한 문제의 경우, 다시 두 가지로 나누어 생각할 수 있다. 일반적인 다이나믹 네트워크 디코더에서처럼 노드와 아크를 네트워크의 관리 단위로 삼을 경우, 작은 크기의 개별적인 구조로 이루어진 이들을 빈번히 생성하거나 철수(withdrawal)하면 메모리는 메모리 끊김(memory fragmentation) 현상을 겪게 된다. 메모리 끊김 현상이 심해지면 운영체제가 지원하거나 시스템에서 사용하는 메모리 관리자(memory manager)가 유효한 메모리 공간을 확보하기 위해 더 많은 계산을 수행해야 하며, 이는 곧 디코딩 시간의 증가를 의미한다.

다른 한가지는 하나의 서브네트워크에 포함되는 노드와 아크를 어떻게 생성하는 가에 대한 문제이다. 즉, 서브네트워크를 생성하거나 철수할 때, 노드/아크들을 개별적으로 처리하게 되면 기존의 동적인 네트워크 관리방법과 결국 동일한 현상

을 겪게 될 것이다. 이 논문에서는 서브네트워크를 "자기 구조화(self-structuring)"된 정적인 표현 방법으로 구성함으로써 이러한 문제를 해결하였다. 자기 구조화된 네트워크란 네트워크가 생성될 때 해당 노드 및 아크를 별도로 생성할 필요 없이 자동으로 구성되도록 고안된 네트워크 표현 방법을 말한다.

<그림 2>(a)는 LM 확률 값을 아크에 분할 저장(factorization) [1][3]한 후의 트리이며 <그림 2>(b)는 자기 구조화된 서브네트워크이다. 노드 및 아크, 그리고 0아닌 아크 가중치들을 각각 노드 집합(node set), 아크 집합(arc set: 아크의 도달 노드 인덱스저장), 가중치 집합(weight set: 아크의 가중치 저장), 외부 집합(extern set: 아크가 다른 서브네트워크로 연결되는 경우, 해당 서브네트워크 인덱스 저장)에 저장하도록 하고 각 원소마다 고유한 인덱스를 부여하였다. 또한 개별 집합들은 서브네트워크의 앞단에 바이트 단위의 거리 정보(offset)를 추가함으로써 쉽게 사용할 수 있도록 하였다. 모든 동질의 요소들은 개별 집합에 포함되기 때문에, 이러한 구성은 메모리 끊김 효과를 방지하게 된다. 한편, 각 노드는 두개의 인덱스 쌍을 포함한다. 한가지는 각 노드의 시작 아크 번호 및 개수이며, 다른 한가지는 가중치 및 외부 집합에 대한 시작 인덱스이다. 아크의 경우, 도착 노드 번호 혹은 외부로 연결되는 서브네트워크의 인덱스가 주어지며, 가중치를 가지는지, 외부 서브네트워크로 연결되는지에 대한 2비트의 플래그를 포함한다. 여기서 주목할 점은 가중치가 없는 아크의 경우 가중치 집합에 포함하지 않는다는 점인데, 이로써 서브네트워크의 크기를 더욱 줄일 수 있었다.



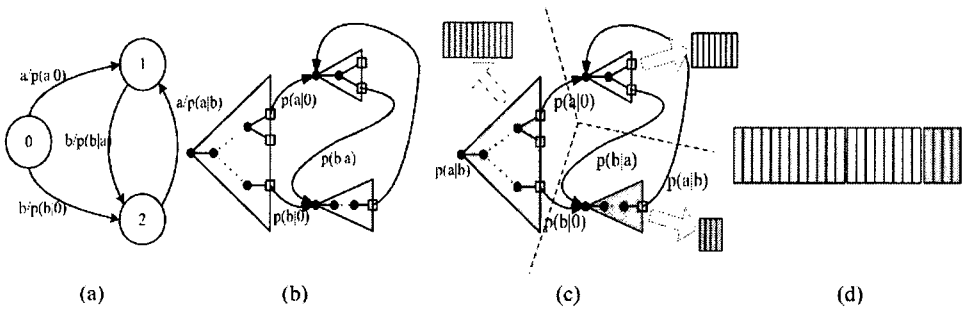
<그림 2> 자기 구조화된 서브네트워크

하나의 트리를 서브네트워크로 변환하는 과정은 일반적인 네트워크 순회(network traversal) 알고리즘을 이용하여 간단히 구현된다. 또한 서브네트워크는 인덱스기반의 표현 방법이기 때문에 노드/아크등의 각 원소들의 실제 주소는 서브네트워크의 실제 주소와 각 집합의 거리로부터 매우 간단하게 계산할 수 있으며, 서브네트워크가 완성되면 그것이 점유하고 있던 메모리영역을 그대로 디스크에 기록하며 디코딩 중에 이를 "그대로" 메모리에 로드하도록 한다. 디코딩 중의 서브

네트워크 생성은 다음에서 볼 수 있듯이 매우 간단히 수행된다.

1. 서브네트워크의 크기만큼 메모리를 할당한다.
2. 디스크로부터 서브네트워크를 읽어온다.
3. 서브네트워크의 앞단에 위치한 각 집합의 거리 정보를 이용하여 각 집합의 실제 주소를 계산한다.
4. 계산된 주소를 집합의 거리가 저장되었던 곳에 다시 저장한다. 이제 모든 원소들은 각 인덱스를 이용하여 사용할 수 있다.

이러한 서브네트워크가 준비되면 전체 탐색 네트워크는 각 후자 트리를 서브네트워크 단위로 변환하는 반복 과정을 통해구할 수 있다. <그림 3>은 이 과정을 보여주고 있다.



<그림 3> 서브네트워크 기반의 탐색 네트워크 구성: (a) 언어 모델 네트워크 (b) 스태틱 네트워크 (c) 서브네트워크 단위 분할 및 변환 (d) 서브네트워크 기반의 탐색 네트워크

4. 세미다이나믹 네트워크 디코딩

4.1. 세미다이나믹 네트워크 관리

논문의 디코더는 토큰 전파 알고리즘 기반의 1-패스 비터비 디코더로 구현되었다. 스태틱 네트워크를 사용하는 경우, 디코더가 할 일은 단순히 토큰을 경로를 따라 전파시키는 일이지만 동적인 관리가 필요한 경우 토큰 전파에 앞서 네트워크의 상태에 따른 적절한 연산이 필요하다. 즉, 토큰이 아직 생성되지 않은 서브네트워크로 전파하려는 경우, 앞 절에서 설명한 방법대로 그것을 생성해야 하며, 더이상 필요하지 않은 서브네트워크는 메모리로부터 철수하도록 해야 한다. 이때 주의할 점은 앞 절에서도 언급했듯이, 서브네트워크의 생성/철수 회수를 최소로

유지하는 일이다. 여기에서는 이러한 생성/철수 연산을 줄이기 위해, 자주 사용되는 서브네트워크들을 캐싱(caching)하거나, 디코딩 이전에 미리 정적으로 로드하도록 하였다.

4.2. 서브네트워크 미리 로드 (subnetwork preloading)

디코딩 중에 자주 활성화되는 서브네트워크들은 동적으로 관리하기보다는 디코딩에 앞서 미리 생성하여 정적인 네트워크로 구성하는 것이 오히려 효율적이다. 여기에서는 다음과 같은 네 가지 종류의 서브네트워크를 미리 생성하도록 한다.

1. 문장 시작(<s>) 및 종료(</s>) 노드에 대응하는 서브네트워크.
2. 문장 시작 다음에 나오는 서브네트워크.
3. 유니그램 (unigram)에 해당하는 서브네트워크.
4. 자주 활성화되리라고 판단되는 서브네트워크.

이중 1-3번은 서브네트워크 "최소 집합"으로 정의한다. 4번의 경우, 각 서브네트워크마다 활성화 빈도수(activation frequency)를 아래와 같이 추정하여 판단하였다.

4.3. 활성화 빈도수 추정

서브네트워크의 활성화 빈도수는 각 서브네트워크와 연관된 언어 모델 히스토리의 확률값(log-likelihood)으로 추정한다. 추정된 값이 p-threshold보다 큰 경우, 또는 상위 n개에 해당하는 서브네트워크를 미리 생성하도록 한다. 가령 두 서브네트워크가 각각 "모자[hat] 를"과 "으로[due to] 결석"이라는 히스토리를 가지고 전자의 확률 값이 후자보다 높다면, 언어 모델의 정의상 전자의 경우가 보다 많이 나타나리라 예상할 수 있으며 이는 연관된 서브네트워크가 보다 빈번히 사용될 수 있음을 의미한다.

언어 모델 히스토리 h에 대한 확률값 계산은 간단하다. 가령 $h = ab$일 때 $p(h) = p(ab) = p(a)p(b|a)$ 을 계산하면 되며, 더욱이 두 확률값 $p(a)$와 $p(b|a)$는 모두 언어 모델 네트워크에 이미 존재하는 값이다. 따라서 언어 모델 네트워크를 만드는 동안 각 노드에 아크의 확률 값을 누적시키면 원하는 계산 값을 얻어낼 수 있다. 즉, 아크의 가중치(언어 모델 확률값)를 그 시점이 되는 노드에 누적된 값에 더한 후 이를 아크의 종점이 되는 노드에 저장하도록 한다. 만약 두 개 이상의 아크가 동일한 노드에 도달하는 경우에는 그 중 최선 값을 유지하도록 한다.

4.4. 서브네트워크 캐싱 (subnetwork caching)

활성화된 토큰을 포함하던 서브네트워크가 모든 토큰을 잃어버리게 되면 해당 서브네트워크는 더 이상 계산할 수 있는 토큰을 포함하지 않으므로 곧바로 메모리로부터 철수시킬 수 있다. (만약 서브네트워크를 버리지 않으면 디코딩에 사용되는 메모리는 지속적으로 늘어날 것이다.) 그러나 잠시 후 토큰이 다시 전파된다면 서브네트워크를 다시 생성해야 하는 번거로움이 발생한다. 따라서 일단 생성된 서브네트워크는 모든 토큰을 잃어버리더라도 일정기간 동안은 다시 활성화될 수 있는 기회를 주는 것이 바람직하다. 일정기간 이후에도 다시 토큰이 전파되지 않는 경우, 구체적으로 말하면 k-threshold 기간(frame)동안 토큰이 전파되지 않을 경우에만 메모리로부터 철수시키도록 한다.

4.5. 서브네트워크 프로파일링 (subnetwork profiling)

서브네트워크 프로파일링 연산은 서브네트워크 미리 로드 연산에 필요한 활성화 빈도수를 보다 정교하게 추정하기 위한 방법이다. 언어 모델 히스토리 정보만을 사용한 5.3절의 방법은 간단하지만, 서브네트워크가 활성화되기 위해서는 언어 모델 뿐 아니라, 입력 음성의 패턴, 음향 모델, 프루닝 방법 등이 종합적으로 결합하여 만들어진 점을 고려할 때, 다분히 제한적일 수밖에 없다. 서브네트워크 프로파일링 과정은 다음과 같이 진행된다. 먼저, 별도의 "프로파일링" 데이터를 준비한다. 이 데이터를 대상으로, "최소집합"에 해당하는 서브네트워크들만 미리 로드한 상태에서 서브네트워크 캐싱 연산 없이(즉, k=0) 디코딩을 진행한다. 디코딩을 진행하는 동안, 디코더는 각 서브네트워크의 활성 횟수를 세도록 한다. 디코딩이 끝나면 각 횟수를 활성화 빈도수로 사용하게 된다. "프로파일링" 데이터는 화자 적응에 사용되는 적응 데이터와 유사한 개념으로, 학습 또는 개발 데이터나, 또는 사용자로부터 별도로 입력받을 수 있다. 프로파일링 데이터가 해당 태스크에 더욱 관련될수록, 관련된 서브네트워크들이 좀더 빈번히 활성화 될 것이기 때문에 보다 정확한 활성화 빈도수를 얻을 수 있을 것이다. 이러한 프로파일링 과정은 세미다이나믹 디코딩을 해당 태스크에 좀더 '맞추어' 적응시키는 과정으로 볼 수 있다.

5. 테일 공유 알고리즘

5.1. 네트워크 최소화 알고리즘

서론에서 언급했듯이, 스태틱 네트워크는 내부의 중복 정보를 제거함으로써 보

다 작은 크기의 동등한 네트워크로 구성할 수 있다. 이를 위한 최적의 방법으로는 모든 상태들 사이의 동등 관계(equivalence relation)을 이용하여 반복적으로 분할해 나가는 오토마타 최소화 알고리즘 및 최소화 알고리즘 적용에 앞서 필요한 결정화(determinization) 알고리즘을 들 수 있다. 그러나 이러한 알고리즘들은 최소화된 네트워크를 만들기 위해 오토마타 상태들 사이에서 만들어진 많은 수의 분할들을 유지해야 하기 때문에, 알고리즘을 적용하기 위해서는 상당한 시간과 메모리가 필요하다. 본 논문의 실험과 같은 대규모 태스크의 경우, 최적화되지 않은 네트워크는 수천만 개까지의 노드들을 포함하게 되는 데, 이러한 경우에는 제한된 계산 자원으로 인하여 그러한 알고리즘들을 실질적으로는 적용할 수 없다는 문제점이 발생한다. 네트워크의 크기를 줄이기 위한 노력가운데 하나로 [10]의 공유 테일을 들 수 있다. [10]에서는 LM확률 값을 분산 적용한 후의 후자 트리의 경우, 동일한 트리의 루트 노드에 연결되는 선형 테일(linear tail)[2]들이 서로 동등한 관계에 있음을 확인하고, 더불어 트리 구조 기반의 탐색 네트워크에 존재하는 대부분의 중복 정보에 해당함을 찾아내어 그러한 테일 들을 공유하도록 하여 근최소화(near-minimized)된 네트워크를 얻을 수 있었다. 한가지 주목할만한 사실은 이러한 공유된 테일 들은 각 후자 트리 마다 만들어진다는 사실이다. 즉, 테일 공유 기법은 동등 관계를 찾기 위해 전체 네트워크를 탐색할 필요 없이, 하나의 후자 트리와 그것에 연결되는 선형 테일들 사이에 "개별적으로" 적용할 수 있는 알고리즘이다. 그러나 원래[10]의 알고리즘은 후자 트리 들이 전체 네트워크로부터 논리적으로 떼내어 다룰 수 없었기 때문에 이러한 개별 적용대신 전체 네트워크를 알고리즘의 입력으로 요구하고 있다. 당연히, 최적화되지 않은 네트워크의 방대한 크기로 인해 알고리즘의 적용은 제한 받을 수밖에 없다. 본 논문의 네트워크 감축 알고리즘은 이러한 테일 공유 알고리즘을 기반으로 다음과 같은 세가지 측면에서 개선되었다.

1. 알고리즘은 서브네트워크를 기반으로 설계되었다. 즉, 알고리즘 적용에 필요한 서브네트워크들만을 메모리에 캐싱하여 전체 네트워크를 메모리에 유지해야 하는 제약 사항을 제거하였다.
2. 원래의 알고리즘은 바이그램 언어 모델을 대상으로 기술되었다. 그러나 본 논문의 경우 N-그램에 제한되지 않는 언어 모델 네트워크를 기반으로 한다.
3. 원래의 알고리즘은 단일 발음 열을 명시적으로 조건화하고 있으나, 본 논문에서는 다중 발음열까지도 명시적으로 처리하도록 하였다. 동일 단어의 다중 발음열 후위(suffix)를 정렬(align)하여 동일한 부분까지도 공유하도록 하여 보다 많은 중복 정보를 제거하였다.

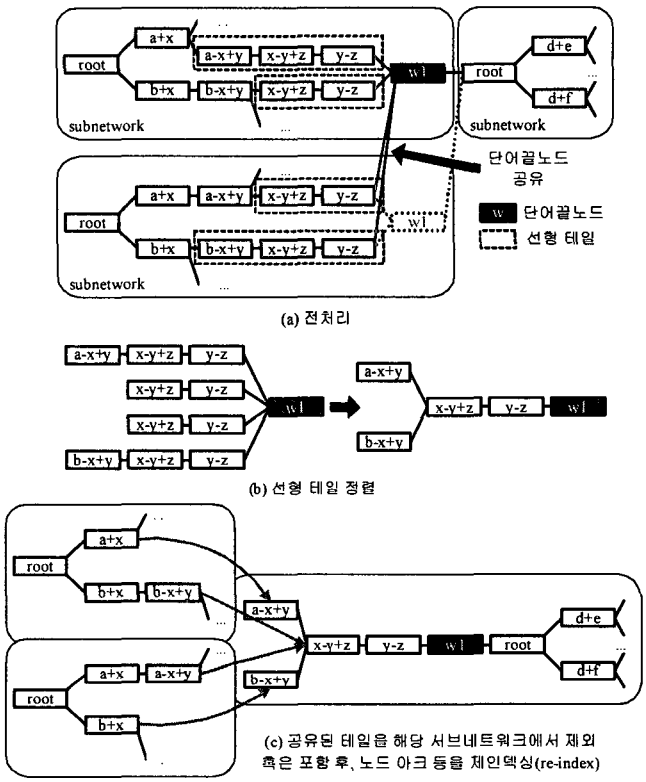
2) 트리의 최하단 노드까지 한 개씩의 자식 노드만을 가지는 경로

5.2. 테일 공유 알고리즘

<그림 4>는 서브네트워크 기반의 테일 공유 알고리즘을 기술하고 있다. 이 알고리즘은 입력으로 서브네트워크기반의 탐색 네트워크를 사용하며, 적용에 앞서 탐색 네트워크를 생성하는 동안 추가적으로인 작업이 필요하다. 먼저 동일한 단어의 마지막임을 알리는 단어끝(word-end)노드[1]를 공유시키고, 선형 테일을 표시해 둔다. 다음, 각 단어끝 노드(we)를 부모 서브네트워크(we.sid)와 노드 인덱스(we.nid), 연결된 선형 테일들을 이용하여 인덱싱을 한다. 모든 전처리가 끝나고 탐색 네트워크가 만들어지면 인덱스된 단어끝 노드를 이용하여 연결된 선형 테일들을 DP (dynamic programming) 알고리즘을 통해 정렬(align)하여 공유된 테일을 만들어 낸다. 이렇게 구성된 공유 테일들은 각기 선형 테일의 부모 서브네트워크들로부터는 제외시키고 루트 노드의 부모 서브네트워크에 포함시키도록 한다. 마지막으로 재구성된 서브네트워크들을 탐색하여 내부의 각 요소(노드, 아크 등)들을 재인덱스하도록 한다. <그림 5>는 테일 공유 알고리즘을 적용하는 과정을 예시하고 있다.

1 단계: 전처리
1. 네트워크 생성시, 1.1 동일 루트 노드에 도달하는 단어끝 노드 공유 및 선형 테일 표시 1.2 단어 끝 노드 **we**를 (**we**.sid, **we**.nid, {**we**에 연결된 선형 테일들})로 인덱스
2 단계: 선형 테일을 정렬하여 테일 공유
2. (1단계에서 인덱스된) 각 단어끝 노드에 대해, 2.1 **we**를 뒤따르는 서브네트워크 **a**에 대해, 공유 테일 **t**를 초기화 2.2 **we**에 연결된 각 선형 테일 **r**에 대해, { 1.2 단계의 결과 이용 } 2.2.1 서브네트워크 **a**의 앞에 위치하며 **r**을 포함하고 있는 서브네트워크 **b**을 파악 2.2.2 **r**을 **t**에 정렬하여 공유 테일로 형성 2.2.3 **b**로부터 **r**을 제외시키고 **r**에 연결되던 아크를 **t**에 정렬된 **r**의 첫 번째 노드에 연결 2.3 **t**를 **a**에 포함시킴
3 단계: 서브네트워크 재 인덱싱(re-indexing)
3. 각 서브네트워크 **s**에 대해, 3.1 네트워크 순회 알고리즘을 이용하여, 노드, 아크, 가중치, 외부 셋을 재인덱싱

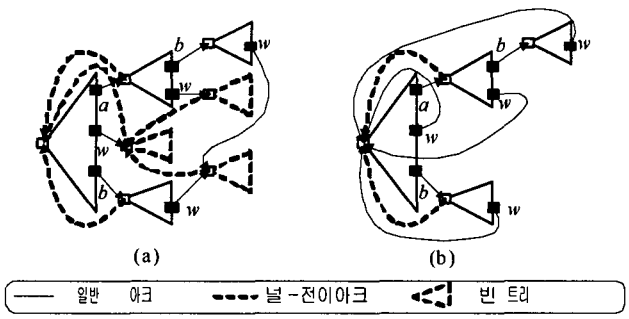
<그림 4> 서브네트워크 기반의 테일 공유 알고리즘



<그림 5> 테일 공유 알고리즘 적용 예제

5.3. 언어 모델 네트워크의 널(null)-전이 아크 제거

N-그램과 같은 통계 모델의 경우, 데이터 부족 현상으로 인해 불안정하게 추정되었다고 생각되는 단어열들에 대한 파라미터들은 종종 잘려(cut-off) 버려지곤 한다. 이러한 경우 다수의 후자 트리들은 후자 단어(successor word)들을 전혀 포함하지 않는 비어있는 트리로 생성되며, 단지 백오프된 히스토리를 갖는 후자 트리로 연결해주는 널-전이(null-transition)만을 수행한다. 이러한 널-전이는 히스토리의 크기가 서로 다른 후자 트리 사이의 테일 공유를 방해하여 네트워크의 감축률을 현저히 떨어뜨리는 요인이 된다. 가령, <그림 6>(a)는 단어 *w*의 출현빈도가 극히 저조하여, *w*로 끝나는 히스토리의 파라미터들이 모두 잘려나간(cut-off) 상황을 보여준다. 그림에서 볼 수 있듯이, 해당 후자 트리들은 모두 백오프 히스토리의 후자 트리로 널-전이만을 수행하며, 선형 테일의 공유도 이루어지지 못하는 것을 알 수 있다. 본 논문에서는 이러한 문제점을 해결하기 위해 빈 후자 트리를 명시적으로 생성하는 대신, <그림 6>(b)와 같이 널-전이 확률값을 이전 연결 아크(preceding arc)에 밀어 넣은 후(weight-pushing) 다음 노드에 직접 연결하도록 하였다.



<그림 6> 널-전이 아크를 제거한 탐색 네트워크

6. 실험 결과

6.1. 실험 환경

본 논문의 시스템을 평가하기 위해 방송 뉴스 인식 태스크를 사용하였다. 음성 DB는 1997년2월부터 1998년 10월까지에 해당하는 KBS 9시 뉴스로부터 앵커 및 기자에 대해 각각 6.5시간, 19시간 분량으로 구성되었다. 텍스트 코퍼스의 경우, 역시 KBS 9시 뉴스의 전사본과 몇몇 신문사의 기사들을 사용하였다. 이들 텍스는 전처리와 형태소 분석 과정을 통해 16M개의 형태소(이 가운데, 52%가 단음절 형태소)를 얻었으며, 이로부터 24k 형태소 및 1,000개의결합 형태소[11]로 이루어진 25k 사전을 사용하였다. 언어 모델의 경우, 수정된 Kneser-Ney 추정 방법을 이용, 빈도수 2회 이하의 이벤트를 컷오프시킨 트라이그램 모델로 3.5M개의 엔트리로 정의되어 있다. 음향 모델은 각 상태당 8개의 가우시안 분포를 가지는 공유-상태 3,557개로 구성한 트라이폰(tied-state triphone) 모델 12,974개를 이용하였다. 테스트를 위해서는 1998년 10월 분량 가운데 OOV가 없는 음성 361 문장(33분정도 분량)을 이용하였다.

6.2. 서브네트워크 기반의 탐색 네트워크 구조 분석

첫 번째 실험으로, 네트워크 감축 기법 사용에 따른 서브네트워크 기반의 탐색 네트워크의 구조 및 크기에 대해 비교 분석하였다. 결과는 네 가지 형태의 네트워크, 즉, 베이스라인 네트워크(3장), NT (null-transition)이 제거된 네트워크(5.3장), 테일 공유된 네트워크(5.2장), NT제거 후 테일 공유된 네트워크들을 대상으로 표 1에 정리하였다. <표 1>에 나타나 있듯이, 최적화되지 않은 베이스라인 네트워크는 1GB가 넘는 상당히 방대한 크기를 가진다. 이러한 방대한 네트워크는 자체로는

별 쓸모가 없겠지만 다음 단계의 네트워크 감축을 위해서는 필요하며, 서브네트워크 기반의 네트워크 구성방법을 이용하여 상당한 크기의 탐색 네트워크라도 생성이 가능함을 알 수 있다. NT제거 알고리즘(5.3절)을 적용한 경우 15.8%, 테일 공유를 한 경우에는 39.1%를 줄일 수 있었으며, 더욱이 이들을 연속 적용한 경우에는 73.4%를 줄일 수 있었다. 네트워크 구성면에서는 베이스라인 네트워크에 비해 NT가 제거된 경우 5분의 1수준의 서브네트워크 만을 필요로 했다. 0아닌 가중치 역시 베이스라인 네트워크의 14.6%, 가장 작은 네트워크의 경우 42.96%로 서브네트워크 생성 시 아크의 가중치를 가려 저장하는 방법이 효과적임을 알 수 있다.

<표 1> 서브네트워크 기반 탐색 네트워크 구조 비교
(%는 베이스라인에 대한 상대 비율)

	베이스라인 (B)	NT-제거 (N)		테일 공유 (T)		NT-제거 및 테일 공유 (NT)	
네트워크 크기 (MB)	1,261.58	1,062.26	(84.2%)	768.33	(60.9%)	335.71	(26.6%)
서브네트워크 개수	2,259,680	485,628	(21.5%)	2,259,680	(100%)	485,628	(21.5%)
노드 개수	44,258,799	40,710,708	(91.9%)	22,365,774	(50.5%)	8,831,930	(19.9%)
아크 개수	53,527,910	49,979,819	(93.4%)	31,787,094	(59.4%)	18,135,303	(33.9%)
가중치 개수	7,814,951	7,814,646		7,791,523		7,791,511	
0아닌 가중치 비율(%)	14.60	15.64		24.51		42.96	

6.3. 디코더 성능

디코더의 인식 성능에 대한 평가는 단어 에러율(WER ; word error rate) 및 프레임당 활성HMM의 평균 개수와 관련하여 <표 2>에 정리하였다. 이 실험에는 1GB의 메모리 및 리눅스가 설치된 2.8GHz 펜티엄 4 PC에서 진행하였다. 탐색 네트워크는 모든 서브네트워크들을 메모리에 로드하여 스태틱 네트워크 디코더로 작동하도록 하였다. 특히 네트워크의 크기가 메모리용량을 넘어서는 B와 N 타입의 네트워크의 경우 다음과 같이 실험을 수행하였다. 즉, 테스트 셋을 세 개로 분할한 후 먼저 프로파일링 연산을 수행하여 디코딩 과정에 사용된 서브네트워크들의 목록을 얻은 후, 다시 이들 서브네트워크들만을 미리 로드하여 성능을 측정하였다. 네트워크의 크기가 큰 경우에도 서브네트워크 연산들이 효과적으로 사용될 수 있음을 보여주는 부분이다. 한편, 5절에 소개된 테일 공유 알고리즘 역시 기본적으로 인식 성능에 영향을 끼치지 않으면서 디코딩의 계산량을 줄여주고 있음을 볼 수 있다. 이러한 계산량 감소는 널-전이 아크들을 제거한 결과이다. 다만 빔크기 120이상의 경우 약간의 성능 향상이 있었는데, 이는 다중 발음열의 후위 부분이 정렬되면서 만들어진 공유 테일에 새로운 네트워크 경로, 즉 새로운 발음열에 해당하는 경로가 만들어졌기 때문이다.

<표 2> 프루닝에 따른 단어에러율(%) 및 탐색 비용 (활성화된 HMM의 평균 개수)

빔크기 (log 단위)		B	N	T	NT
100	WER	22.92	22.92	22.92	22.92
	#Avg	1547.62	1534.16	1463.22	1417.65
110	WER	21.21	21.21	21.21	21.21
	#Avg	2748.83	2720.79	2759.62	2471.88
120	WER	20.34	20.34	20.32	20.32
	#Avg	4636.43	4581.68	4311.32	4078.76
130	WER	20.35	20.35	20.33	20.33
	#Avg	7588.18	7485.02	6982.36	6506.35
140	WER	20.15	20.15	20.13	20.13
	#Avg	12043.5	11858.2	10960.0	10037.0
150	WER	20.06	20.06	20.04	20.04
	#Avg	18638.0	18367.9	16815.2	10657.9

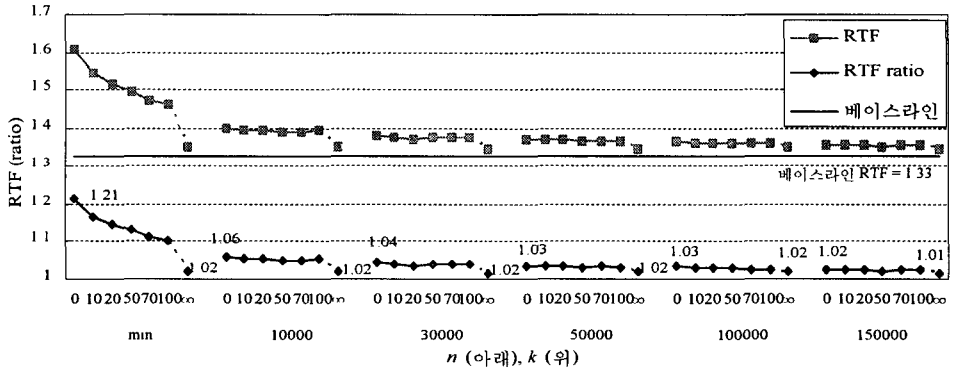
이하 실험에서는 NT 타입의 네트워크 및 프루닝 크기 120을 베이스라인으로 사용하였다. 이 경우 활성화된 HMM의 최대 개수는 146,689개이며, 스태틱 네트워크 디코딩시 1.33 RTF의 인식 시간이 소요되었다.

6.4. 서브네트워크 캐싱 성능

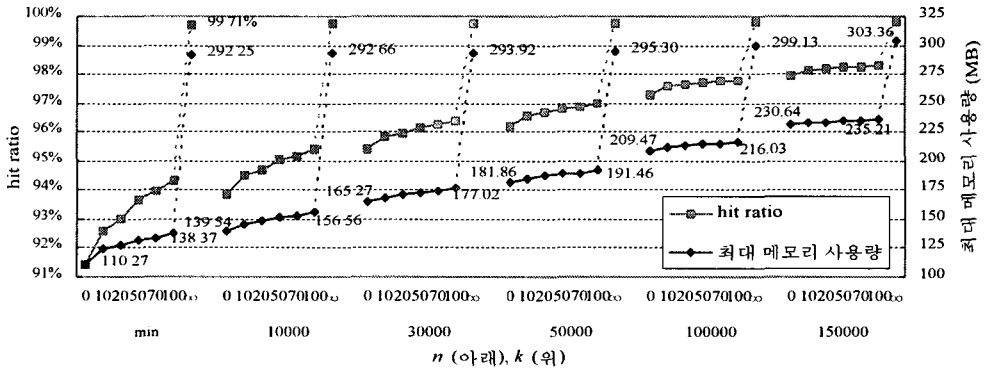
이 절에서는 서브네트워크 캐싱의 성능에 대해 정리하도록 한다. 캐싱은 적은 메모리를 이용하여 서브네트워크들을 가능한 한 빠르게 사용하도록 하는 데에 목적이 있으므로, 캐싱 성능은 메모리상의 서브네트워크에 대한 H/R(hit ratio), 메모리 사용량(혹은, 사용된 서브네트워크 개수), 인식시간 비율인 RTF 비율(RTF ratio)[3] 등으로 파악할 수 있다. <그림 7>(a)는 캐싱 파라미터(n 및 k)에 따른 RTF 및 RTF ratio를 보여주고 있다. "최소집합"의 서브네트워크들만을 미리 로드하고(n=min) 캐싱 하지 않은 경우(k=0), 최대 메모리 사용량은 전체 네트워크 크기의 약 3분의 1수준인 110MB의 메모리를 사용한 반면 RTF는 21%가 증가함을 볼 수 있다. 그러나 일단 캐싱이 사용된 후에는 이 RTF ratio는 10%이내로 떨어져서, 10k

3) RTF ratio는 각 RTF를 베이스라인의 RTF로 나누어 계산한다.

의 서브네트워크를 미리 로드하는 경우 추가적으로 30MB정도의 메모리로 RTF ratio를 6%수준으로 떨어뜨릴 수 있다. 한편, 캐시된 서브네트워크를 디코딩 하는 동안 전혀 해제하지 않는 경우(k=8), 99.71% 이상의 상당히 높은 hit ratio를 얻을 수 있으나, 메모리 사용량 역시 300MB안팎에 이르며 더구나 디코딩 시간의 감소는 상대적으로 그다지 눈에 띄지 않음을 볼 수 있다. 이러한 결과는 단기적으로 볼 때 대부분의 디코딩 계산이 전체 탐색 네트워크 가운데 지역적으로 몰려있는 반면 (활성화된 탐색 공간), 장기적으로 볼 경우 그러한 활성화된 탐색 공간이 넓게 자리잡고 있음을 보여준다. 실험결과로부터 서브네트워크 미리 로드 및 캐싱 연산은 이러한 탐색 공간의 양면성에 잘 대처하고 있음을 알 수 있다. 즉, 서브네트워크 캐싱 연산은 현재 탐색이 진행되는 활성화된 탐색 공간에 초점을 두고 있으며, 반면 서브네트워크 미리 로드 연산은 전역적으로 걸쳐있는 가능한 탐색 공간들을 결정하는 데 유용하게 사용될 수 있다.



(a) RTF ratio



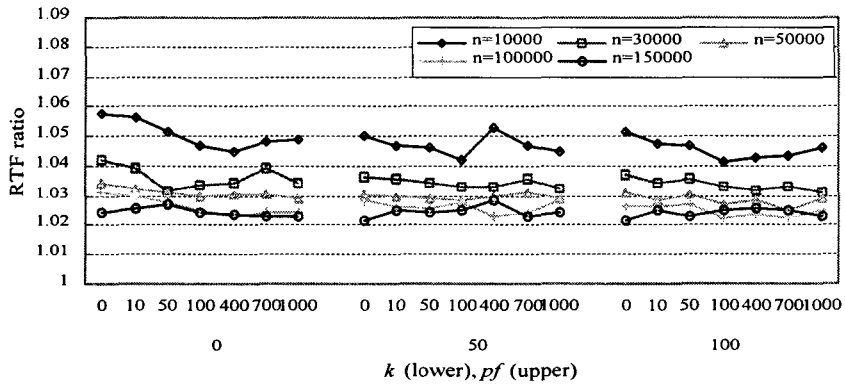
(b) 최대 메모리 사용량

<그림 7> 서브네트워크 캐싱 성능

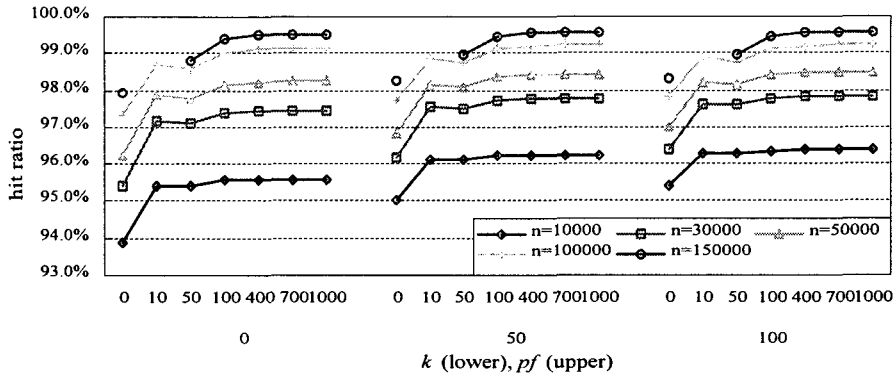
6.5. 프로파일링을 통한 서브네트워크 캐싱 성능 향상

<그림 8>는 서브네트워크 프로파일링 과정을 통해 향상된 캐싱 성능을 보여주고 있다. 10~1000까지의 *pf*값은 프로파일링에 사용된 문장 수를 말하며 *pf*=0은 프로파일링을 하지 않은 결과, 즉, 앞 절의 결과를 가리킨다. 프로파일링 데이터의 경우, 학습 데이터로부터 테스트 데이터의 바로 앞 쪽 날짜부터 점진적으로 포함시켜 가는 식으로 구성하였다. 즉, *pf*=50의 프로파일링 데이터는 *pf*=10의 프로파일링 데이터 이외에 10문장 앞에 나오는 40개의 문장을 추가하여 구성하였다.

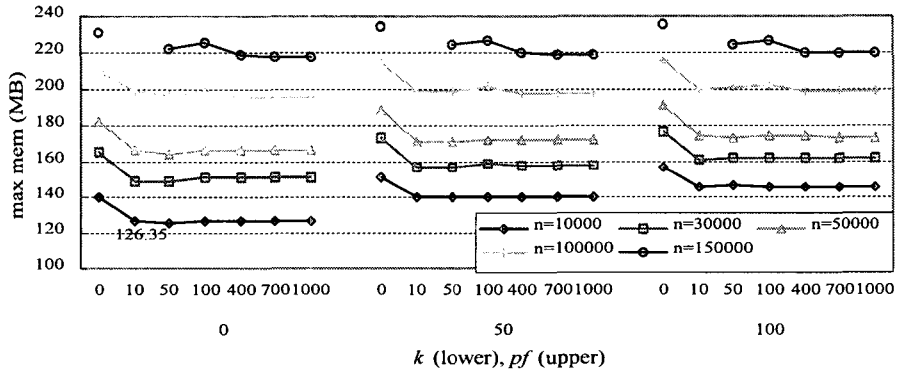
프로파일링을 하지 않은 결과와 비교해 볼 때, hit ratio와 최대 메모리 사용량은 상당한 성능향상이 있었음을 볼 수 있었다. RTF ratio의 경우, 10,000개의 서브네트워크를 미리 로드한 경우에는 눈에 띄는 향상이 있었으나, 더 많은 부분을 미리 로드한 경우의 결과는 프로파일링을 하지 않은 결과와 큰 차이가 없음을 볼 수 있다. 구체적으로, *pf*=10, *n*=10,000을 사용하는 경우, 프로파일링을 하지 않은 경우에는 캐싱을 위해 30MB의 추가 메모리가 필요하였으나 프로파일링을 거친 후에는 절반에 가까운 16MB의 추가 메모리만을 필요로 하였다. (네트워크 크기의 37.6% 수준) 한편 프로파일링 과정은 일종의 데이터 기반의 추정 방법이므로 미리 로드되는 규모에 비해 너무 적은 양의 프로파일링 데이터를 사용하는 경우, 그래프의 *n*=100,000인 부분에서 확인할 수 있듯이 불안정한 결과를 얻거나, *n*=150,000 및 *pf*=10에서처럼 전혀 추정치를 얻을 수 없는 상황(이 경우, 활성화된 서브네트워크의 수는 110,000개가 채 되지 못하였다.)에 이르게 되기도 한다.



(a) RTF ratio 측면



(b) Hit ratio 측면



(c) 최대 메모리 사용량 측면

<그림 8> 프로파일링 과정을 통한 캐싱 성능 향상

7. 결 론

이 논문에서는 서브네트워크 기반의 탐색 네트워크 구성 방법과 토큰 전파 알고리즘을 이용한 세미다이나믹 네트워크 디코더를 소개하였다. 이 디코더는 기존의 정적 네트워크 디코더의 장점인 디코더 자체의 효율성과 동적 네트워크 디코더의 장점인 메모리 효율성을 동시에 살릴 수 있었다. 탐색 네트워크의 특성을 반영하고 디코딩에 효율적인 서브네트워크 기반의 탐색 네트워크를 구성하는 방법을 제안하였으며, 이러한 서브네트워크 구성 방법을 기반으로, 서브네트워크 기반의 테일 공유 알고리즘을 개발, 효과적으로 네트워크의 중복 정보를 제거하고, 인식 성능의 저하 없이 메모리 사용량 및 계산량을 상당히 줄일 수 있는 방법을 소

개하였다. 이러한 새로운 디코딩 방법들은 방송 뉴스 인식 실험을 통해 대규모의 태스크에도 효과적으로 적용될 수 있음을 보여주었다. 추후 연구 과제로는 현재 새로운 네트워크 구성 방안으로 제시되고 있는 WFST (weighted finite-state transducer)기반의 알고리즘과 상호 운영하는 방법에 관한 연구가 진행되어야 한다.

참 고 문 헌

[1] 안동훈, 정민화, "언어 모델 네트워크에 기반한 대어휘 연속 음성 인식", *한국음향학회지*, 제21권 6호, 2002년 8월.

[2] D.-H. Ahn and M. Chung, "Finite State Network Decoder Based on Language Model Network", *Proc. of International Conference on Speech Processing*, pp.1015-1019, Taejon, Korea, Aug. 2001.

[3] G. Antoniel, F. Brugnara, M. Cettolo, and M. Federico, "Language Model Representations for Beam-search Decoding", *Proc. of 1995 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp.588-591, Detroit, MI, USA, May 1995.

[4] M. Mohri and M. Riley, "Integrated Context-dependent Networks in Very Large Vocabulary Speech Recognition", *Proc. of 6th Europoean Conference on Speech Communication and Technology*, pp.811-814, Budapest, Hungary, Sep, 1999.

[5] H. Ney, R. Haeb-Umbach, B.H. Tran and M. Oerder, "Improvements in Beam Search for 10000-word Continuous Speech Recognition", *Proc. of 1992 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp.9-12, San Francisco, CA, USA, Mar. 1992.

[6] P.C. Woodland, C.J. Leggetter, J.J. Odell, V. Valtchev and S.J. Young, "The 1994 HTK Large Vocabulary Speech Recognition System", *Proc. of 1995 IEEE International Conference on Acoustics, Speech and Signal Processing*, Detroit, MI, USA, May 1995.

[7] S. Ortmanns, H. Ney. "Look-Ahead Techniques for Fast Beam Search", *Computer, Speech and Language*, vol.14, no.1, pp.15-32, Jan. 2000.

[8] F. Alleva, X. Huang, and M.-Y. Hwang, "Improvements on the Pronunciation Prefix Tree Search Organization", *Proc. of 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp.133-136, Atlanta, GA, USA, May 1996.

[9] M. Mohri, F. Pereira, and M Riley, "Weighted Finite-state Transducers in Speech Recognition," *Computer Speech and Language*, vol.16, no.1, pp.69-88, Jan. 2002.

[10] F. Brugnara and M. Cettolo, "Improvements in Tree-based Language Model Representation", *Proc. of 6th Europoean Conference on Speech Communication and Technology*, pp.2133-2136, Madrid, Spain, Sep. 1995.

[11] Y.-H. Park, D.-H. Ahn, and M. Chung, "Morpheme-based Lexical Modeling for Korean Broadcast News Transcription", *Proc. of 8th Europoean Conference on Speech Communication and Technology*, pp.1129-1132, Geneva, Switzerland, Sep. 2003.

접수일자: 2004년 2월 17일
게재결정: 2004년 6월 7일

▶ 정민화(Minhwa Chung)
주소: 121-742 서울시 마포구 신수동 1 서강대학교
소속: 서강대학교 컴퓨터학과
전화: 02) 712-8023
E-mail: mchung@sogang.ac.kr

▶ 안동훈(Dong-Hoon Ahn)
주소: 121-742 서울시 마포구 신수동 1 서강대학교
소속: 서강대학교 컴퓨터학과
전화: 02) 712-8023
E-mail: drahn@nlpeng.sogang.ac.kr