J. Inf. Technol. Appl. Manag. 23(2): 61~80, June 2016
http://dx.doi.org/10.21219/jitam.2016.23.2.061

ISSN 1598-6284 (Print)
ISSN 2508-1209 (Online)

제약식 프로그래밍을 이용한 일방향 전송 무선 메쉬 네트워크에서의 최적 링크 스케쥴링

김 학 진*

The Optimal Link Scheduling in Half-Duplex Wireless Mesh Networks Using the Constraint Programming

Hak-Jin Kim*

Abstract

The wireless mesh network (WMN) is a next-generation technology for data networking that has the advantage in cost and the flexibility in its construction because of not requiring the infra-structure such as the ethernet. This paper focuses on the optimal link scheduling problem under the wireless mesh network to effectuate real-time streaming by using the constraint programming. In particular, Under the limitation of half-duplex transmission in wireless nodes, this paper proposes a solution method to minimize the makespan in scheduling packet transmission from wireless nodes to the gateway in a WMN with no packet transmission conflicts due to the half-duplex transmission. It discusses the conflicts in packet transmission and deduces the condition of feasible schedules, which defines the model for the constraint programming. Finally it comparatively shows and discusses the results using two constraint programming solvers, Gecode and the IBM ILOG CP solver.

Keywords : Link Scheduling, Wireless Mesh Network, The Constraint Programming, TDMA

Received : 2016. 03. 23. Revised : 2016. 06. 03. Final Acceptance : 2016. 06. 07.

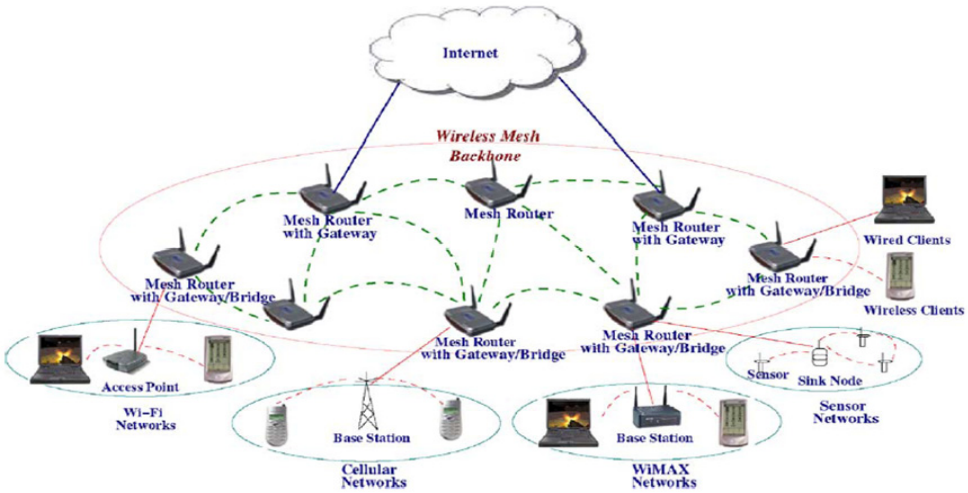
* School of Business, Yonsei University, #415, Yonsei University, 50 Yonsei-ro Seodaemun-gu, Seoul, 03722 Korea, Tel : +82-2-2123-2497, e-mail : hakjin@yonsei.ac.kr

1. 서 론

최근 십여 년간 무선 네트워크 기술은 비약적인 발전을 이루었다. 특히 WiFi 구축의 비용상 기술상의 어려움으로 저렴한 비용의 무선 애드혹 네트워크(wireless ad hoc network, WANET) 기술이 각광을 받게 되었다. 여러 애드혹 네트워크 기술 중 무선 메쉬 네트워크(Wireless Mesh Network, WMN) 기술은 유선과 비슷한 전송 속도로까지 발전한 무선 통신 기술과 함께 기존의 이더넷 같은 기반 시설에 의존하지 않고 무선 라디오 노드들 간의 연결성을 이용하여 기기 간에 동적으로 네트워크를 구성하는 유연성을 갖는다. 사전에 네트워크를 위한 인프라를 고정적으로 설계할 필요 없이 상황에 따라 유연하게 기기의 위치 변동을 통하여 네트워크를 상황에 적합하게 변경할 수 있다. 따라서 이를 이용하여 무선 백본망을 구축할 시 네트워크의 신뢰성과 확장성 그리고 이동성 등의 여러 장점을 갖게 된다.

<Figure 1>은 메쉬 라우터와 메쉬 클라이언트로 이루어진 네트워크 망이 인터넷과 연결되어 백본망을 형성하는 예를 보여주고 있다. 이는 더 나아가 기존의 인터넷, 와이파이, 와이맥스, 센서 네트워크 그리고 이동통신망들과 연결되어 혼합된 형태의 무선 메쉬 네트워크를 구축할 수도 있다[Wei et al., 2005]. 이 기술을 통하여 응용될 수 있는 분야는 홈 네트워킹, 지역 사회 네트워킹, 기업 네트워킹, 도심 네트워킹, 교통시스템 네트워킹, 빌딩 자동화 네트워킹, 의료 및 건강 시스템 네트워킹, 보안 감시 네트워킹 등 많은 영역에서 사용이 가능하다[Akyildiz et al., 2005].

이와 같은 차세대 네트워크를 효과적으로 사용하기 위하여 여러 가지 연구되어야 할 이슈들이 있는데, 그 중 무선 메쉬 네트워크의 동적 연결성으로 인하여 데이터를 전송하기 위해서는 어떻게 링크 간의 전송 스케쥴을 만들 것인가 하는 문제는 중요한 문제이다[Shetiya and Sharma, 2005]. 특히 링크의 연결성이 라디오 주파수에 의존하여 네트워크 노드 간의 간섭 및 지연 현상 등이 발생할 수 있고 이로 인하여 전체 네트워크 망의 효율이 떨어질 수 있는 무선 메쉬 네트워크의 특이성 때문에 이를 완화 혹은 회피할 수 있는 방식의 링크 전송 스케쥴의 고안은 무선 메쉬 네트워크의 효과적 운영에 매우 중요한 사안이다.



source : Akyildiz et al.[2005].

〈Figure 1〉 An Example of WMN Backbone

이를 위하여 일반적으로 시분할 다중연결(Time Division Multiple Access, TDMA) 방식의 스케쥴링이 주된 연구 분석의 틀이 된다[Djukic and Valaee, 2007]. 이는 무선채널의 주파수 선택 문제를 고려하고 OFDM 변조 기법 등의 사용으로 TDMA 방식이 현실에 부합하기 때문이다[Djukic and Valaee, 2009]. 즉 스케쥴링 하고자 하는 시간 프레임을 여러 개의 작은 시간 분할 슬롯으로 표현하고 각 시간 슬롯에 네트워크의 어떤 노드에 어떤 패킷 전송을 할당하는지를 결정하는 문제로 귀결시킨다. 그리고 이와 같은 패킷 전송 할당의 문제에서 WMN이 갖고 있는 특이점으로부터 발생하는 여러 문제를 회피하도록 전송 할당 방법을 고안하는 것이 주 목적이 된다.

이 논문은 WMN 기반 하의 여러 응용에서 적용할 수 있는 실시간 패킷 전송을 목적으로 하는 최적의 링크 스케쥴링을 구하는 것을 그 목표로 하고 있다. 특히 무선 노드들 간에 패킷의 전송과 수신이 동시에 이루어지지 못하는 일방향 전송(halt-duplex) 상황을 반영한 패킷 전송 스케쥴링을 목표로 한다. 이를 위해 TDMA를 이용한 WMN 스케쥴링 구조를 통해 링크의 패킷 전송에 시간 슬롯을 할당하는 문제로 본다. 이는 보통 일방향 전송을 고려하지 않을 때 Djukic and Valaee[2007]의 주장에서와 같이 WMN에서 문제를 야기하지 않는 스케쥴을 그래프로 표현하고 이 위에서 그래프 엣지(edge)의 색깔을 정하는 엣지 컬러링(edge coloring) 문제로 치환할 수 있다. 하지만 이는 이 논문에서 다루고자 하는 WMN의 일방향 전송 상황의 문제를 용이하게 다루지 못하는 한계를 지닌다. 이는 이 특수한 경우에 엣지 컬러링과 같은 분할(partitioning) 이상의 복잡한 조합 구조(combinatorial structure)를 내포하기 때문이고 단순 TDMA 스케쥴링 문제 이상의 복잡성을 갖게 된다. 이와 같은 복잡한 문제의 해결을 위해 Kim[2013]은 해밀토니언 사이클(Hamiltonian cycle)[Diestel, 1997; Vandegriend, 1998]을 이용한 스케쥴링 휴리스틱을 제안하였지만 이 역시 완전성에서 떨어진다. 따라서 이 논문은 일방향 전송 상황에서의 최적의 링크 스케쥴을 구하기 위한 방법으로서 인공지능 분야의 제약식 프로그래밍 기법을 제안한다. 먼저 주어진 상황으로부터 패킷 전송 스케쥴의 충돌 현상에 대해 분석하고 이로부터 실현가능조건을 도출한다. 얻어진 실현가능조건을 이용해 스케쥴링 문제를 풀기 위한 제약식 프로그램 모형을 구축한다. 구축된 모형은 제약식 프로그래밍 솔버로서 잘 알려진 Gecode와 IBM ILOG CP의 두 가지 솔버를 이용하여 해를 구해보고 이 두 솔버의 결과를 가지고 제기된 문제의 해결 성과를 비교 분석해보고자 한다.

이를 위하여 논문은 다음과 같이 구성된다. 이 서론 뒤에 제 2장으로 무선 메쉬 네트워크에서의 링크 스케쥴링 문제에 대한 관련 연구를 제시하고 현재까지의 연구에 대해 소개한다. 제 3장에서 이 논문에서 해결하고자 하는 문제에 대한 정의와 스케쥴 충돌에 대한 문제를 논의한다. 제 4장에서는 충돌 문제에서 스케쥴의 실현가능조건을 도출하고 이를 통해 문제 해결을 위한 제약식 프로그래밍 모형을 제시한다. 또 여기서는 구체적인 문제들을 Gecode와 ILOG CP를 이용하므로 이 솔버들에서 문제를 표현하기 위한 구체적 모형 구축에 필요한 논의를 진행한다. 제 5장에서는 계산 실험을 위하여 구체적인 문제의 인스턴스의 생성에 대해 논의하고 생성된 문제 인스턴스를 이용한 Gecode와 ILOG CP 모형의 구축 그리고 최적해를 구하는 문제 풀이 시간 비용에 대한 측정을 제시함으로써 두 솔버를 이용한 풀이에 대한 비교를 한다. 마지막으로 결론으로 이 연구의 의의에 대한 논의를 하고 연구의 한계점과 앞으로 연구 이슈들에 대한 제안을 한다.

2. 관련 연구

무선 메쉬 네트워크의 성능에서 가장 중요한 요소 기술중 하나는 MAC 스케줄러이다. 무선 메쉬 네트워크의 스케줄링은 중앙 집중형과 분산형 두 가지로 구분할 수 있다[Ghazvini et al., 2010]. 중앙 집중형 스케줄링의 경우 게이트웨이 노드가 각 메쉬 노드에게 고정 시간 슬롯을 비율적으로 할당하며, 분산형 스케줄링의 경우 제어 메시지 전송을 관리하는 스케줄링 기능과 데이터 영역 할당을 관리하는 스케줄링 기능으로 구분된다. 메쉬 네트워크 표준화 규격에서는 두 가지의 스케줄링 방식에 대한 세부적인 알고리즘을 정의하지 않았으며 구현 이슈로 열어 두었다. 본 절에서는 두 가지 스케줄링에 대한 세부적인 알고리즘 측면에서의 이슈를 정리하며 최근 연구 동향에 대해서 연구하였다.

무선 메쉬 네트워크의 스케줄링에는 시분할 다중연결(Time Division Multiple Access, TDMA) 방식이 주로 이용된다. 일반적인 무선 네트워크에서 사용하는 CSMA/CA MAC 프로토콜을 사용할 경우 무선 메쉬 네트워크에서 요구하는 QoS를 보장하지 못하는 경향이 있기 때문이다[Kim, 2013]. TDMA의 경우 다른 방식들, 주파수분할 다중연결(Frequency Division Multiple Access, FDMA)이나 코드분할 다중연결(Code Division Multiple Access, CDMA)과 다르게 무선 채널의 주파수 선택성과 OFDM 같은 변조기법의 사용으로 좀 더 실용적으로 선택되는 방식이다[Gore and Karandikar, 2011]. TDMA는 전체 전송 시간을 단위 시간의 슬롯으로 분할하여 네트워크 각 노드에서의 전송을 서로 다른 시간 슬롯에 발생하도록 함으로써 전송 간에 벌어질 수 있는 충돌 현상을 회피하고 전체 네트워크의 밴드위드(bandwidth)를 효과적으로 사용하도록 해 준다. 따라서 이 논문에서도 TDMA를 이용한 경우를 상정하여 논의한다. 특히 네트워크에서의 노드의 패킷 전송을 시간 슬롯의 연속적인 할당으로 보는 것 대신, 노드 간의 간섭현상을 회피하는 조건에서 서로 직접적인 관련인 없는 노드들 상에 동시에 전송이 이루어지는 공간 시분할 다중연결(Spatial Time Division Multiple Access, STDMA) 방식을 전제로 한다.

Gore and Karandikar[2011]의 논문은 TDMA를 기반으로 멀티홉 무선 네트워크(multi-hop wireless networks)에서의 링크 스케쥴의 문제를 다루었다. 특히 네트워크상의 링크 간의 충돌에서 발생할 수 있는 지체를 최소화하는 방향으로 패킷 스케쥴링을 하는 휴리스틱 알고리즘을 제안하고 있다. 먼저 TDMA에서의 네 가지 형태의 충돌을 규정하고 이에 따라 충돌 그래프를 설정한다. 그리고 한 노드에 연결된 수신 링크 위의 발신 시간과 발신 링크 위의 수신 시간의 차이를 지체로 보았다. 그래서 링크들의 순서가 정해졌을 때 주어진 슬롯의 수 N에 대해 충돌을 방지하면서 왕복 이동 경로(round-trip path) 상에서 지체를 줄이는 스케쥴을 찾는 문제를 최소-최대(min-max) 정수 계획 문제로 설정하였고, 문제의 복잡성을 줄이고자 오버레이 트리(overlay tree) 토폴로지 경우로 제한하여 이를 최소 경로 문제(the shortest path problem)로 해결하는 휴리스틱을 제시하였다. 이 휴리스틱을 이용해 실현 가능 스케쥴이 존재하는 최소의 N을 규정하고, 이를 패킷 프레임의 슬롯과 비교하여 충돌 그래프의 정보에 따라 링크의 순서를 수정하였다. 마지막으로 수정된 링크 순서에 따라 스케쥴을 수정해 주는 방식으로 최종 스케쥴을 결정하였다. 제시된 방법은 무선 네트워크상에서의 링크 스케쥴에 대한 지체를 정의하고 이를 이용해 충돌을 방지하는 스케쥴을 보다 합리적으로 제시하였다는데 의미가 있다. 하지만 이 논문도 일반적인 무선 네트워크를 상정하였고 네트워크 노드 상에 버퍼가 존재하는 경우를 다룸으로써 이 논문에서 상

정하는 문제와 차이가 있다. 또한 이 논문에서 제시한 방법은 근본 개념에서 부터 버퍼가 있는 상태를 상정하고 그 휴리스틱의 구조가 복잡하여 제시된 수정하는 방법으로는 이 논문에서 제시하는 문제를 해결하기 어렵다고 판단된다.

Salem and Hubaux[2005]의 논문은 무선 메쉬 네트워크에서 어떻게 하면 각 클라이언트가 요구하는 패킷의 전송을 공정하게 다루도록 스케쥴 하는지에 대해 관심을 기울이고 있다. 무엇보다도 이 논문에서 다루는 네트워크의 상황이 이 논문에서 다루는 상황과 유사하다. 즉, 무선 메쉬 네트워크를 인터넷과 연결해 주는 게이트웨이 혹은 핫스팟이 존재하고 여러 개의 AP가 네트워크를 구성하고 있다. 그리고 각 AP에는 패킷의 전송을 요구하는 클라이언트 디바이스들이 연결된다. 문제에서 네트워크의 토폴로지는 고정되어 있다고 가정한다. 이 때 각 클라이언트가 요구하는 패킷 전송을 공정하게 처리하기 위하여 각 클라이언트 별 전송 처리량(throughput)이 동일하다는 조건에서 C를 각 링크의 최대 처리량, T를 스케쥴의 한 사이클 시간이라 보고 이를 C/T로 계산한다. 이는 결국 네트워크의 총 처리량을 최대화하기 위해 T를 최소화하는 문제로 연결된다. 최적에 가까운 스케쥴을 구하기 위해 저자는 먼저 링크 간의 간섭과 의존성을 갖지 않는 관계를 보완행렬(compatibility matrix)로 표현하고 이를 노드행렬(adjacency matrix)로 간주한 그래프에서 클릭(clique)을 도출한다. 한 클릭에 포함된 링크들을 같은 시간슬롯에 배정함으로써 간섭현상을 회피하게 되고 최적해에 가까운 스케쥴을 찾기 위해 그리디 방법(greedy algorithm)을 사용한다. 하지만 이 경우의 문제는 각 AP가 버퍼를 지닌 것을 가정하고 있다. 따라서 버퍼가 없는 경우 한 라우트의 연속된 링크에서 연속적으로 패킷을 전달하기 위해서는 클릭 간의 선후관계에 제약을 갖게 되고 또한 두 개의 클릭에 포함된 인접한 한 쌍의 링크가 연속적으로 배정된다고 할 때 이 클릭에 포함된 다른 쌍의 인접한 링크가 연속적으로 배정되는 것을 보장하지 못하는 근본적인 문제가 내제되어, 제안된 방식을 일부 수정하는 방식으로는 이 논문에서 제기하는 문제를 해결하기 힘들 것으로 판단된다.

Gore and Karandikar[2011]의 논문은 무선 메쉬 네트워크에서의 링크 알고리즘에 대해 문헌 연구를 한 논문으로 STDMA 관점에서의 패킷 전송을 그래프로 묘사하고, 이 위에 STDMA 관점의 패킷 전송이 조합 최적화의 에지 컬러링(edge coloring) 모형으로 구현할 수 있음을 보였다. 즉 같은 시간 슬롯에서 전송이 이루어지는 노드 간의 연결 링크들에 해당하는 에지에 같은 색깔을 부여하는 것이다. 기본적으로 에지 컬러링 문제는 NP-hard, 좀 더 정확하게 근사-PTAS(Approximate Polynomial Time Approximate Scheme)에 속하는 문제로 이를 풀 수 있는 다항식 알고리즘이 알려져 있지 않아 근사 알고리즘의 방식을 통해서 문제를 해결한다[Ausiello et al., 1999]. Gore and Karandikar[2011]는 STDMA에서의 링크 스케쥴 문제를 에지 컬러링의 틀에서 보면서 무선 메쉬 네트워크에서 실제 발생할 수 있는 간섭 현상을 고려하여 세 가지 모형으로 기존의 연구들을 분석하였다. 각 모형별로 제시된 알고리즘은 기본적으로 휴리스틱으로 최적의 스케쥴을 제시하지는 못하지만 주어진 간섭 현상을 회피할 수 있는 기준에 따라 표현 그래프의 에지를 컬러링함으로써 링크 스케쥴의 실현 가능 해를 제시하고 있다. 하지만 본 연구에 관심을 갖는 문제를 직접적으로 해결하는 데에는 한계점을 갖는 것으로 보인다. 기본적으로 무선 메쉬 네트워크의 링크 스케쥴링을 보는 틀을 제시함으로써 이를 다양하게 응용할 수 있는 여지를 주었지만, 이 연구에서는 패킷이 단순히 네트워크상에 흘러갈 때 노드들의 전송, 수신, 대기 등의 상태에

따른 일반적인 스케쥴이 아니라 하나의 게이크웨이 노드가 존재하여 이에 모든 패킷이 이미 형성된 네트워크를 통해 수집되어야 하는 상황이고 각 노드마다 버퍼가 부재함으로써 고려해야하는 제약으로 말미암아 단순 적용은 무리가 있다.

이상의 최근 논문들은 문제의 단순화를 위해 각 무선 노드들이 양방향 전송(full-duplex)이 가능하다는 전제하에 스케쥴링 문제를 고려하고 있어 무선 노드의 일반적인 일방향 전송(half-duplex) 성격을 반영하지 못한다. 이를 반영한 논문으로서 Kim[2013]은 패킷의 전송에서 발생하는 충돌의 경우들의 패턴들을 제시하고 문제를 해결하기 위한 휴리스틱으로서 해밀토니안 사이클 알고리즘에서 얻어진 스케쥴을 미리 분석된 충돌 패턴을 회피하기 위하여 공 슬롯을 삽입하는 방식으로 패킷 전송 스케쥴을 만든다. 하지만 이 논문은 충돌 패턴들을 완전하게 탐색되지 못하였고 제안한 휴리스틱도 임시방편적인 조작으로 근본적인 해결과는 거리가 있다. 즉 단순히 해밀토니안 사이클에 의존함으로써 조합적인 구조(combinatorial structure)상 문제가 분할(partition)과 순서(sequence)의 두 가지 속성을 함께 지님에도 도출되는 해의 경우 순서 만을 고려하기 때문으로 생각된다. 따라서 제시된 문제의 해결을 위해 이 논문에서는 제시된 문제의 구조적인 부분을 파악하여 이에 대한 완전한 제약식 프로그래밍 모형을 제시하고 최적의 스케쥴을 구할 수 있는 풀이 방식을 보여주고자 한다. 특히 이 논문에서 제시되는 방법은 단순 실현가능해를 찾는 휴리스틱이 아닌 최적해를 구하는 방법을 제시한다.

제약식 프로그래밍(The Contstraint Programming)은 인공지능의 한 분야로서 여러 응용 분야들에서 발생하는 문제들을 제약식 만족 문제(Constraint Satisfaction)로 표현하고, 연역 논리와 계산 기법들을 이용하여 시스템적으로 구현 실제 문제들의 해결기를 구축하는 프로그래밍 개념을 일컫는다. 문제 표현을 위한 프로그래밍의 기본 개념을 제약식(Constraint)으로 보고 응용문제의 도메인에 따라 표현을 위한 변수와 이 변수들 간의 관계를 적합한 제약식을 통해 해결하고자 하는 문제를 정의함으로써 모형화 한다. 그리고 문제의 도메인 상에서 규정된 제약식의 특수한 성질을 이용하여 제약식을 만족하는 변수들의 값을 도출하거나 탐색을 이용한 일반적인 방법을 통해 문제의 해를 도출한다[Apt, 2003; Dechter, 2003; Van Hentenryck, 1989]. 여기서는 패킷의 스케쥴을 표현하는 변수들의 도메인을 정수로 보고 기존의 정수 제약식 프로그래밍인 한정도메인 제약식 프로그래밍(Finite-Domain Constraint Programming)으로 문제를 모형화하여 한정도메인 제약식 프로그래밍 해결기인 Gecode와 ILOG CP를 이용하여 문제를 해결하고자 한다[Van Hentenryck, 1999; Schulte et al., 2015].

3. 문제의 정의

3.1 패킷 전송 네트워크와 문제의 상황

이 연구에서 고려하는 네트워크는 외부의 인터넷이나 유선망에 연결되는 한 개의 게이트웨이 노드와 이 게이트웨이에 의해 커버되는 지역의 다수의 메쉬 라우터로 구성된다. 문헌에서 네트워크에 가해지는 로드를 완화시키고 밸런스를 이루기 위해 다수의 게이트웨이 노드를 사용하는 연구가 있지만 이는 게이트웨이 노드 한 개의 경우에 대한 확장으로 볼 수 있으므로 게이트웨이 노드 한 개에 대한 연구가 기본이 된다고 볼 수 있다. 또 네트워크에서 발생하는 패킷의 전송은 업스트림과 다운스트림의 두 종류가 있을 수 있지만 네트워크 패킷 전송 스케쥴링의 기본적인 성격 상 업스트림을 가정한다. 즉 각 메쉬 라우터에서는 게이트웨이 노드로 보낼 업스트림 패킷을 발생시키고 이를 네트워크를 통해 게이트

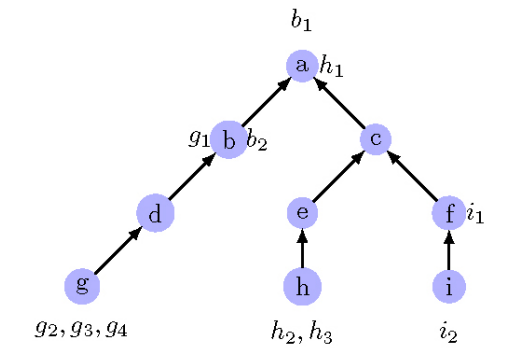
웨이 노드까지 성공적으로 전달함을 목표로 한다. 이를 위해 스케쥴링 의사결정 전에 각 메쉬 라우터는 전달해야 할 패킷의 양과 위치에 대한 정보를 게이트웨이 노드로 전달하고 게이트웨이는 이 정보를 바탕으로 네트워크 토폴로지를 결정함으로써 패킷 전송 스케쥴링 의사결정을 위한 네트워크의 구조를 결정한다. 네트워크 토폴로지의 형성은 여러 가지 상황을 고려한 여러 방법들을 생각할 수 있지만, 가장 간단하게 게이트웨이 노드와 메쉬 라우터의 거리와 메쉬 라우터들 간의 거리를 고려하여 최단거리 경로 알고리즘에 의해 만들어진 네트워크 트리를 생각해 볼 수 있다. 본 연구에서의 관심은 이와 같이 게이트웨이 노드와 메쉬 라우터 간의 네트워크 토폴로지가 트리 구조로 결정된 후 각 메쉬 라우터부터 게이트웨이 노드까지의 시간별 패킷 전송을 계획하는 스케쥴링 문제이다. 즉 어떤 메쉬 라우터로부터 어떤 시간슬롯에 패킷을 전송하여 이미 만들어진 네트워크 트리를 거쳐 게이트웨이 노드에 최단시간 안에 전달하도록 하느냐 하는 것이다. 여기서 TDMA 기반의 스케쥴링을 전제하므로 TDMA의 공간 재활용(spatial reuse) 방식을 적용하여 고려 대상인 노드와 직접적인 링크관계를 갖지 않는 다른 노드들도 이전 시간에 수신한 패킷을 동시에 전송할 수 있음을 가정한다. 그리고 이 때 패킷 전송 스케쥴은 메쉬 라우터의 기능적인 제약으로 말미암아 다음과 같은 조건을 만족한다.

메쉬 라우터는 수신된 패킷을 다음 시간에 전송하는 단순한 역할만을 수행한다. 자체적으로 스케쥴 상의 의사결정을 하는 계산 모듈은 존재하지 않는 것을 전제한다.

게이트웨이 노드를 제외한 메쉬 라우터는 전방향 안테나(omni-directional antenna)를 사용하고 이에 따라 신호를 동 시간에 전송(send)과 수신(receive)을 할 수 없는 일방향 전송(half-duplex) 성격을 지니다.

3.2 패킷 충돌과 회피

앞에서 설명된 라우터의 기능적인 제약으로 인해 패킷 전송 스케쥴 상의 어려움이 나타나게 된다. 조건 1)에 따르면 패킷의 방출(release)이 시작되어 다른 노드에 전달되었을 때 이 패킷은 노드에 저장됨 없이 곧바로 게이트웨이 노드에 가까운 네트워크상의 상위 노드에 전달되어야 한다. 또 조건 2)로 말미암아 어느 한 노드가 패킷을 받아서 상위 노드로 전달할 때까지 다른 노드들은 그 노드로 또 다른 패킷을 전달할 수 없으며, 그 패킷은 조건 1)에 의해 노드에 대기 상태로 있을 수도 없다. 따라서 문제에서의 어려움은 이와 같은 충돌 현상이 발생하지 않도록 모든 패킷의 방출이 계획되어야 한다는 것이다. <Figure 2>는 패킷 충돌의 한 예를 보여주고 있다. <Figure 2>에서 노드 b, g와 h 그리고 i에서 패킷이 b_1, b_2와 g_1, g_2, g_3, g_4와 h_1, h_2, h_3 그리고 i_1, i_2가 전송되어야 한다고 할 때, 이 네 노드에서 b_1, h_1, g_1 그리고 i_1이 연속으로 방출되어 5단위 시간 뒤의 상황을 묘사하고 있다. 그림의 아래 부분은 패킷 방출에 대한 스케쥴을 시간 슬롯에 표시하고 있다. 그림 상 충돌이 발생하는 곳은 노드 b로 여기에 패킷 g_1이 위치하는데 스케쥴 상 b_2를 방출하게 되어 패킷 충돌이 발생하고 있다.



b_1	h_1	g_1	i_1	b_2	h_2	g_2	i_2	b_3	h_3	g_3	$\emptyset$	g_4	$\emptyset$

source : Kim[2013].

〈Figure 2〉 An Example of Packet Confliction

〈Table 1〉 Insertion of Blank Slots to Avoid Packet Confliction

Height gap	Type of path	Gateway crossing	General node crossing
0		xyx	x0y0x
1	same	xy0x	xy00x
	different	xy0x	xy00x
2	same	xy00x	xy000x
	different	xyx	xy000x
3	same	xy000x	xy0000x
	different	xy0x	xy0000x
4	same	xy0x	xy00000x
	different	xyx	xy00000x
5	same	xy00x	xy00x
	different	xy00x	xy00x
6	same	xy00x(xy000x)	xy000x
	different	xyx	xy000x
7	same	xy0x(xy000x)	xy000x(xy0000x)
	different	xy0x	xy000x(xy0000x)
8	same	xy000x(xy0000x)	xy0000x(xy00000x)
	different	xyx	xy0000x(xy00000x)

source : Kim[2013].

Kim[2013]은 두 개의 패킷이 발신되는 위치의 게이트웨이 노드를 기준으로 상대적인 높이차, 경로의 상이성, 패킷 경로 상 교차 노드의 형태에 따라 충돌의 패턴을 확인하였다. 그리고 패킷 전송을 위한 스케쥴링을 위하여 해밀토니안 경로를 이용한 알고리즘으로 전송 스케쥴을 생성할 때, 이를 통해서 위에서 설명한 패킷 충돌 문제가 나타나는 현상을 막을 수 없으므로 이를 방지하기 위해 방편적인 방법으로 시간 슬롯에 공 슬롯을 끼워 넣는 방식으로 스케쥴을 수정하는 방식을 사용하였다. Kim[2013]은 패킷 충돌을 방지하기 위하여 <Table 1>과 같은 공 슬롯 삽입 패턴을 제시하였다. 하지만 위에서는 모든 충돌 패턴들이 다 제시되지 못하고 제시한 휴리스틱의 성능에 대한 분석도 부족하였다.

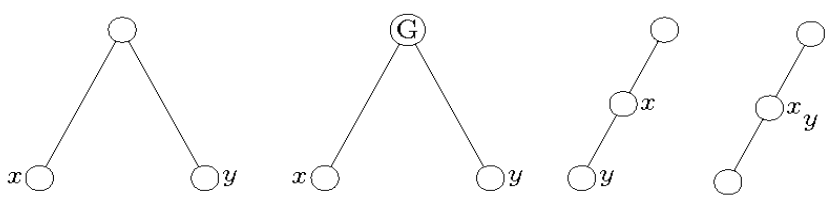
Kim[2013]의 논문에서는 이와 같은 충돌 패턴을 모두 탐구하여 제시하지 못하였지만 이 논문에서는 제시되지 못한 패턴을 포함, 모든 충돌 패턴을 방지하기 위해 충족되어야 하는 조건을 제시한다.

4. 제약식 프로그래밍을 이용한 모델링

4.1 실현가능 스케쥴 조건

여기서는 앞 장에서처럼 패킷 전송에서 나타나는 충돌의 경우를 열거하지 않고, 충돌의 원인을 파악하기 위한 패턴을 분석하였다. 그 패턴은 <Figure 3>에서와 같이 네 가지 경우로 정리된다. 그리고 이와 같이 분석된 충돌 패턴을 방지하기 위한 패킷 전송 스케쥴의 실현 가능 조건을 제시하였다. 즉 제시된 실현 가능 조건을 충족하는 패킷 전송 스케쥴을 작성하였을 시 이는 <Figure 3> 제시된 패킷 충돌 패턴을 갖지 않게 되며 따라서 앞 장의 모든 충돌 패턴들도 나타나지 않게 된다.

게이트웨이 노드로의 업스트림을 간주하고 그 방향이 위인 경우를 상정하였을 때, 첫 번째 경우는 두 개의 패킷 x와 y가 경로 상 일반 노드에서 충돌하는 경우이다. <Figure 3>에서 보는 것처럼 상위의 일반 노드는 두 개의 패킷을 동시에 받을 수 없다. 두 번째는 상위의 교차 노드가 게이트웨이인 경우로 이때에도 마찬가지로 두 개



〈Figure 3〉 Four Kinds of Packet Confliction

의 패킷 x와 y을 동시에 받을 수 없다. 셋째로는 패킷 x가 중간 노드에 위치하고 있는 상태에서 이 노드는 새로운 패킷 y의 수신과 현재 가지고 있는 패킷 x의 전송을 동시에 수행할 수 없다. 넷째 경우는 중간에 있는 노드가 현재 패킷 x을 수신하여 가지고 있는 상태에서 노드 자체의 패킷 y를 생성하여 발신할 수 없다. 기본적으로 Kim[2013]의 논문에서 관찰한 패킷 충돌의 경우는 결국 이 네 가지 중의 한 가지 상태로 귀결되어 충돌이 나타나게 되며 이 네 가지 경우는 네트워크의 정의 상 나타난 노드들의 기능상의 한계와 직접적으로 연결되어 설명된다. 즉 첫째와 둘째는 노드가 동시에 수신을 할 수 없는 한계, 셋째는 수신과 전송이 동시에 발생할 수 없는 한계, 넷째는 기본적으로 노드에서 한 개의 패킷만을 가질 수 있는 버퍼 부재의 한계이다.

이들 네 가지 충돌 형태로부터 실현 가능 스케쥴이 갖추어야 하는 패킷 전송의 규칙을 다음과 같이 요약할 수 있다.

가) 두 개의 패킷은 동시에 한 링크의 말단 노드에 위치할 수 없다.
나) 각 노드에는 오직 한 개의 패킷만이 위치할 수 있다.

우선 가)와 나)의 조건을 만족한다면 앞에서 제시한 네 가지 패킷 충돌의 형태를 회피할 수 있다. 즉 여기서 버퍼부재의 한계를 유지하기로 하면 패킷은 패킷의 방출과 동시에 연속적으로 네트워크를 따라 이동해야 한다. 이를 가정하고 노드 상 패킷 전송의 한계를 제거하였을 때, 만약 패킷의 전송 형태가 첫째와 둘째의 충돌 경우라면 전송 스케쥴에 따라 두 개의 패킷이 동시에 상위 노드로 올 수 있다. 이는 나) 규칙을 깨게 된다. 셋째와 넷째의 경우는 곧바로 가) 규칙에 대한 위반으로 나타난다. 따라서 위 두 개의 규칙을 지키게 된다면 앞의 네 가지 충돌의 경우는 발생하지 않게 된다. 마찬가지로 가)와 나)의 규칙의 위반을 허용한다면 앞의 네 가지 충돌의 경우가 발생할 수 있어 앞의 충돌 회피가 불가능하게 된다. 따라서 버퍼 부재의 한계는 패킷의 방출 이후 네트워크상의 게이트웨이 노드를 향한 경로를 따라 패킷의 연속적인 전송 스케쥴로 표현되고 만약 위에 제기한 규칙을 지키는 조건 하에서 그 스케쥴을 계획할 수 있다면 원하는 충돌을 회피하는 스케쥴을 완성할 수 있게 된다. 이를 계획하기 위한 모형을 다음에 제시한다.

4.2 제약식 프로그래밍 모형의 설정

여기서는 앞의 두 개의 실현가능 스케쥴의 조건을 이용하여 제약식 프로그래밍 모형을 구축한다. 우선 여기서 문제를 단순화하기 위하여 각 노드에는 전송하고자 하는 패킷이 하나씩 있다고 가정한다. 이 가정은 문제의 일반성을 훼손하지 않는다. 왜냐하면 한 노드에 여러 개의 전송하고자 하는 패킷이 존재하는 경우는 이를 그 노드에 또 하나의 연속적인 추가 노드들로 연결된 가지를 설정하고 각 노드에 원 노드에서 보내고자 하는 패킷을 배치시키면 동일한 효과를 거둘 수 있다.

다음 모형에서는 두 가지의 변수를 사용한다. $T_{p,v}$는 주어진 트리에서 생성된 패킷 p가 트리의 노드 v에 도달했을 때의 시간을 나타내고 T는 모든 패킷이 싱크로 전달되어 전송 작업이 마쳐진 시점을 나타낸다. 이 변수 정의를 이용하여 다음의 제약식들이 설정된다.

4.2.1 도메인 제약식(In-Domain Constraints)

$$T_{p,v} \in \{1, \cdots, T_{\max}\} \qquad \forall p, v \tag{1}$$

$$T \in \{1, \cdots, T_{\max}\} \tag{2}$$

한정 도메인 제약식 프로그래밍에서 모든 변

수는 일차적으로 해의 후보 값으로 한정된 도메인을 갖는 것으로 설정된다. 위의 제약식에서 각 변수들은 값의 후보로 1에서 $T_{\max}$값 사이에 있는 것으로 설정된다. $T_{\max}$는 각 변수가 취할 수 있는 가장 늦은 시점으로 임의의 큰 값이 될 수 있지만, 기본적으로 문제에서 주어진 정보를 이용하여 나타날 수 있는 가장 큰 값으로 설정하면 된다.

4.2.2 한계 제약식(Bound Constraints)

$$T_{p,v} \leq T \qquad \forall p, v \qquad (3)$$

스케쥴링 상 모든 전송이 완료되는 시간인 T는 모든 패킷 p가 노드 v에 위치하는 시간 $T_{p,v}$ 중에 가장 최종 시간으로 결정된다. 따라서 위와 같은 제약식이 성립한다. 또 목표로 T 값을 최소화함으로 위의 제약식과 결합하여 T는 최종시간으로 결정된다.

4.2.3 거리 제약식(Distance Constraints)

$$T_{p,v} = T_{p,s(v)} + 1 \qquad \forall p, v \qquad (4)$$

여기서 $s(v)$는 트리의 뿌리 노드로부터 각 최종 노드로 이어지는 줄기에 따라 순서를 정했을 때 노드 v의 후속 노드이다. 따라서 위의 제약식은 패킷 p가 노드 v에 도착하는 시간은 후속 노드에 도착한 시간에서 1단위 시간 이후가 됨을 의미한다.

4.2.4 노드 제약식(Node Constraints)

트리의 각 노드 v에 대하여 다음과 같은 제약식을 설정한다.

$$AllDifferent(T_{p,v}) \quad \forall p \in P(\tau(v)) \qquad (5)$$

먼저 전체 트리에서 임의의 노드 v에 대해 v를 포함한 v의 하부트리 $\tau(v)$를 정하면 여기에 포함된 모든 노드와 여기서 생성된 패킷에 대하여 $AllDifferent$ 제약식을 적용한다. 이 제약식의 의미는 관련된 모든 변수의 값이 모두 다름을 의미하는 것으로, 즉 하부트리에서 생성된 패킷이 지정된 노드 v에 도착할 때의 시간은 모두 다른 시간이 되어야 함을 뜻한다.

4.2.5 링크 제약식(Link Constraints)

트리의 각 링크 $e=(u, v)$에 대하여 다음과 같은 제약식을 설정한다. 이때 $v=s(u)$를 전제한다.

$$T_{p,u} \neq T_{q,v}, \quad \forall p \in P(\tau(u)), q \in P(\tau(v)), p \neq q \qquad (6)$$

여기서 보듯이 p는 노드 u의 하부트리에서 생성되는 패킷이고 q는 노드 v의 하부트리에서 생성되는 패킷이다. 앞에서 $v=s(u)$가 가정되므로 당연히 $\tau(u) \supset \tau(v)$의 식이 성립하고 p는 $P(\tau(v))$에 있는 패킷일 수도 있다. 하지만 p는 q와는 다른 패킷을 나타낸다. 따라서 위 식은 u와 v의 각 하부트리의 패킷이 전송되어 링크 e에 도달되었을 때, 동 시간대에 링크의 말단 노드에 위치할 수 없음을 나타낸다. 왜냐하면 동 시간에 위치함은 다음 시간대에 노드 u에서의 송신과 수신이 함께 이루어져야 함을 나타내고, 만약 함께 이루어지지 않아 노드 v로부터의 수신이 뒤늦게 이루어진다면 노드 v에 버퍼가 존재하여야 하기 때문이다.

4.2.6 목적식(Objective)

$$\min T \qquad (7)$$

앞의 제약식을 만족하는 많은 $T_{p,v}$의 값의 대안들 중에서 최종 시간인 T를 최소화하는 해를 찾는 것이 여기서 고려하는 스케쥴링 문제이다.

4.3 두 가지 제약식 프로그래밍 솔버에 대한 모형 언어의 고려

이 논문에서는 제약식 프로그래밍 솔버로서 ILOG CP와 Gecode를 사용하여 문제를 푼다. 일반적으로 스케쥴링 문제는 정수계획법 같은 최적화 방법론을 택할 수 있으나 $AllDifferent$나 $\neq$를 이용한 제약식을 직접적으로 표현할 수 있는 방법이 정수계획법에는 없다. 이를 다수의 부등식과 이접 제약식(disjunctive constraints)를 이용하여 표현하는 방법을 생각해 볼 수 있으나 제약식 프로그래밍의 제약식에 대응하는 부등식의 개수가 지수 함수적으로 증가하여 실제적으로 불가능하다고 볼 수 있다. 그리고 제약식 프로그래밍 문제를 푸는 솔버로서 ILOG CP는 상업적으로 가장 성공한 솔버이고, Gecode는 연구 영역에서 최근까지 개발된 여러 구현 중 가장 속도가 빠른 것으로 알려져 있다. 따라서 두 솔버를 이용해서 문제를 푸는 것 외에도 아직 비교 연구가 이루어지지 않은 두 솔버를 이 연구의 문제 풀이를 통하여 간접 비교해 보는 것도 의미가 있기에 이 두 솔버가 채택되었다.

```
using CP;
int maxTime = ...;
tuple Index {
  string Packet;
  int    Node;
};
{Index} ScheduleIndices = {
  <“a”,0>, <“a”,1>,
  <“b”,0>, <“b”,1>, <“b”,2>,
  <“c”,0>, <“c”,1>, <“c”,3>,
  <“d”,0>, <“d”,1>, <“d”,3>, <“d”,4>,
  <“e”,0>, <“e”,1>, <“e”,3>, <“e”,5>
};
dvar int t[ScheduleIndices] in 0..maxTime;
dvar int T in 0..maxTime;
minimize T;
subject to {
  t[<“a”,0>]<=T; t[<“a”,1>]<=T;
  t[<“b”,0>]<=T; t[<“b”,1>]<=T; t[<“b”,2>]<=T;
  t[<“c”,0>]<=T; t[<“c”,1>]<=T; t[<“c”,3>]<=T;
  t[<“d”,0>]<=T; t[<“d”,1>]<=T; t[<“d”,3>]<=T; t[<“d”,4>]<=T;
  t[<“e”,0>]<=T; t[<“e”,1>]<=T; t[<“e”,3>]<=T; t[<“e”,5>]<=T;
  t[<“a”,0>]==t[<“a”,1>]+1;
  t[<“b”,0>]==t[<“b”,1>]+1; t[<“b”,1>]==t[<“b”,2>]+1;
  t[<“c”,0>]==t[<“c”,1>]+1; t[<“c”,1>]==t[<“c”,3>]+1;
  t[<“d”,0>]==t[<“d”,1>]+1; t[<“d”,1>]==t[<“d”,3>]+1; t[<“d”,3>]==t[<“d”,4>]+1;
  t[<“e”,0>]==t[<“e”,1>]+1; t[<“e”,1>]==t[<“e”,3>]+1; t[<“e”,3>]==t[<“e”,5>]+1;
  allDifferent( all (i in {<“a”,0>,<“b”,0>,<“c”,0>,<“d”,0>,<“e”,0>}) t[i] );
  allDifferent( all (i in {<“a”,1>,<“b”,1>,<“c”,1>,<“d”,1>,<“e”,1>}) t[i] );
  allDifferent( all (i in {<“c”,3>,<“d”,3>,<“e”,3>}) t[i] );
  t[<“a”,1>]!=t[<“b”,2>]; t[<“c”,1>]!=t[<“b”,2>]; t[<“d”,1>]!=t[<“b”,2>];
  t[<“e”,1>]!=t[<“b”,2>];
  t[<“a”,1>]!=t[<“c”,3>]; t[<“a”,1>]!=t[<“d”,3>]; t[<“a”,1>]!=t[<“e”,3>];
  t[<“b”,1>]!=t[<“c”,3>]; t[<“b”,1>]!=t[<“d”,3>]; t[<“b”,1>]!=t[<“e”,3>];
  t[<“c”,1>]!=t[<“d”,3>]; t[<“c”,1>]!=t[<“e”,3>]; t[<“d”,1>]!=t[<“c”,3>];
  t[<“d”,1>]!=t[<“e”,3>]; t[<“e”,1>]!=t[<“c”,3>]; t[<“e”,1>]!=t[<“d”,3>];
  t[<“c”,3>]!=t[<“d”,4>]; t[<“e”,3>]!=t[<“d”,4>];
  t[<“c”,3>]!=t[<“e”,5>]; t[<“d”,3>]!=t[<“e”,5>];
};
```

〈Figure 4〉 Example of ILOG CP Models

불행히도 이 두 가지 솔버가 다룰 수 있는 제약식의 형태나 표현 문법에서 차이가 있어 각 솔버에 맞는 두 가지 표현의 다른 솔버 모형을 사용해야 한다. 먼저 ILOG CP의 경우는 일반적으로 IBM Cplex Optimization Studio의 OPL 모형 언어를 사용한다. 뒤에 무작위로 생성된 문제를 풀 때 OPL을 이용하여 앞 절에서 정의한 모형을 표현하게 된다. <Figure 4>는 이렇게 만들어진 OPL 모형을 설명하고 있다.

ILOG CP를 위한 OPL 언어가 선언적인 속성을 지닌 반면 Gecode는 프로시져 언어의 속성을 지닌 C++로 구현된 관계로 입력 모형에 있어서 선언적이라기보다는 과정적인 속성을 지닌다. <Figure 5>는 Gecode에서 풀게 되는 인스턴스의 한 예를 보여주고 있는데 C++의 클래스 선언을 연상시키는 형태로 모형이 표현되고 있다. Gecode의 모형 표현에서는 식 (5)의 *AllDifferent* 제약식을 distinct 제약식으로 표현하고 있다. 특히 ILOG CP에서의 *AllDifferent* 제약식이 한정 도메인 변수들만으로 정의되는데 반하여 Gecode의 제약식은 다음과 같은 형태를 허용한다.

distinct(home, c, x); (8)

이는 한정 도메인 변수들의 벡터 x에 대해 상수 벡터 c가 관련되어 다음과 같은 관계가 성립된다.[Gecode Manual]

$$x_i + c_i \neq x_j + c_j \qquad 0 \leq i,j \leq |x|, i \neq j$$

이 제약식의 확장성으로 Gecode의 모형에서 식 (4)를 생략하고 제약식의 c를 통해 distinct 제약식을 정의하게 된다. 근본적으로 ILOG CP의 *AllDifferent* 제약식은 distinct 제약식과는 다르게 상수의 추가를 허용하지 않고 변수로만 구성되도록 되어 있다. 이 제약은 어느 한 노드에 도달한 패킷의 도달 시점을 변수로 지정함을 요구하고 이를 위하여 식 (4)에서와 같이 패킷 이동에 따른 각 노드의 도달 시점을 변수화하고 제약식을 통해 계산할 필요가 있다. 하지만 Gecode에서는 distinct 제약식의 확장성으로 각 패킷의 첫 생성 시기만을 변수로 정의하고 다른 노드에 도달 시기는 상수 c을 통하여 표현할 수 있게 된다. 노드에서 식 (4)는 즉 노드 u에 대한 distinct 제약식을 정의할 때 노드 u의 하부트리 에 있는 서로 다른 노드 v와 w에서 생성된 패킷 p와 q가 노드 u에 도달할 때의 시간은 다른 시점이 되어야 하고, 이는 노드 v와 w에서 노드 u까지의 거리를 각각 d_v와 d_w라 할 때 다음과 같은 식이 성립해야 한다.

$$T_{p,v} + d_v \neq T_{q,w} + d_w \qquad \forall v,\ w \in \tau(u),\ v \neq w$$

위 식에서 $T_{p,v}$는 패킷 p가 노드 v에서 생성되는 시각이므로 d_v를 더함으로써 패킷 p가 노드 u에 도달하는 시각을 나타낸다.

두 번째로 고려할 사항은 마찬가지로 식 (4)의 불필요성으로 인해 식 (6)의 제약식을 다음과 같은 형태로 나타내게 된다. 그리고 이는 Gecode의 linear 제약식으로 구현된다.

$$T_{p,x} + d_x \neq T_{q,y} + d_y$$
$$\forall x \in \tau(u),\ y \in \tau(v),\ x \neq y,\ p = p(x),\ q = p(y)$$

링크 $e = (u, v)$에 대해 하부트리의 두 개의 다른 노드 x와 y에서 생성된 패킷 p와 q가 링크의 말단에 동시에 도달하는 것을 막기 위하여 d_x와 d_y를 각각 노드 u와 v까지의 거리라고 하고 식의 좌변과 우변을 노드 u와 v에 도달한 시각을 표현하여 이들이 다름을 나타낸다.

결국 두 솔버의 모형은 제약식의 확장성의 차이에 의하여 다수의 변수의 설정 여부에서 차이가 발생하게 되는데 계산의 측면에서 변수의 개수가 적은 편이 유리하지만 그 차이는 클 것으로

여겨지지는 않는다. 왜냐하면 제약식 프로그래밍의 제약식 탐색법(constraint propagation)으로 패킷이 경유하는 노드에서의 시점을 나타내는 변수 중 하나의 값이 고정됨과 동시에 다른 경유 노드의 시점들도 큰 노력 없이 고정될 수 있기 때문이다.

```
...
class StreamSchedule : public IntMinimizeScript {
protected:
  int size;
  IntVarArray t;
  IntVar finish;
public:
  StreamSchedule(const Options& opt) : size(6), t(*this, size, 0, 72), finish(t[0]) {
    IntArgs c(size);
    for (int i=0; i<size; i++) c[i]=0; c[0]=1; c[1]=-1; linear( *this, c, t, IRT_GQ, 1 );
    for (int i=0; i<size; i++) c[i]=0; c[0]=1; c[2]=-1; linear( *this, c, t, IRT_GQ, 2 );
    for (int i=0; i<size; i++) c[i]=0; c[0]=1; c[3]=-1; linear( *this, c, t, IRT_GQ, 2 );
    for (int i=0; i<size; i++) c[i]=0; c[0]=1; c[4]=-1; linear( *this, c, t, IRT_GQ, 3 );
    for (int i=0; i<size; i++) c[i]=0; c[0]=1; c[5]=-1; linear( *this, c, t, IRT_GQ, 3 );
    IntVarArgs n0; IntArgs d0;
    n0 << t[1] << t[2] << t[3] << t[4] << t[5]; d0 << 1 << 2 << 2 << 3 << 3;
    distinct(*this, d0, n0, ICL_BND);
    IntVarArgs n1; IntArgs d1;
    n1 << t[1] << t[2] << t[3] << t[4] << t[5]; d1 << 0 << 1 << 1 << 2 << 2;
    distinct(*this, d1, n1, ICL_BND);
    IntVarArgs n3; IntArgs d3;
    n3 << t[3] << t[4] << t[5]; d3 << 0 << 1 << 1; distinct(*this, d3, n3, ICL_BND);
    for (int i=0; i<size; i++) c[i]=0; c[1]=1; c[2]=-1; linear(*this, c, t, IRT_NQ, 0);
    for (int i=0; i<size; i++) c[i]=0; c[3]=1; c[2]=-1; linear(*this, c, t, IRT_NQ, -1);
    for (int i=0; i<size; i++) c[i]=0; c[4]=1; c[2]=-1; linear(*this, c, t, IRT_NQ, -2);
    for (int i=0; i<size; i++) c[i]=0; c[5]=1; c[2]=-1; linear(*this, c, t, IRT_NQ, -2);
    for (int i=0; i<size; i++) c[i]=0; c[1]=1; c[3]=-1; linear(*this, c, t, IRT_NQ, 0);
    for (int i=0; i<size; i++) c[i]=0; c[1]=1; c[4]=-1; linear(*this, c, t, IRT_NQ, 1);
    for (int i=0; i<size; i++) c[i]=0; c[1]=1; c[5]=-1; linear(*this, c, t, IRT_NQ, 1);
    for (int i=0; i<size; i++) c[i]=0; c[2]=1; c[3]=-1; linear(*this, c, t, IRT_NQ, -1);
    for (int i=0; i<size; i++) c[i]=0; c[2]=1; c[4]=-1; linear(*this, c, t, IRT_NQ, 0);
    for (int i=0; i<size; i++) c[i]=0; c[2]=1; c[5]=-1; linear(*this, c, t, IRT_NQ, 0);
    for (int i=0; i<size; i++) c[i]=0; c[3]=1; c[4]=-1; linear(*this, c, t, IRT_NQ, 0);
    for (int i=0; i<size; i++) c[i]=0; c[3]=1; c[5]=-1; linear(*this, c, t, IRT_NQ, 0);
    for (int i=0; i<size; i++) c[i]=0; c[4]=1; c[3]=-1; linear(*this, c, t, IRT_NQ, -2);
    for (int i=0; i<size; i++) c[i]=0; c[4]=1; c[5]=-1; linear(*this, c, t, IRT_NQ, -1);
    for (int i=0; i<size; i++) c[i]=0; c[5]=1; c[3]=-1; linear(*this, c, t, IRT_NQ, -2);
    for (int i=0; i<size; i++) c[i]=0; c[5]=1; c[4]=-1; linear(*this, c, t, IRT_NQ, -1);
    for (int i=0; i<size; i++) c[i]=0; c[3]=1; c[4]=-1; linear(*this, c, t, IRT_NQ, 0);
    for (int i=0; i<size; i++) c[i]=0; c[5]=1; c[4]=-1; linear(*this, c, t, IRT_NQ, -1);
    for (int i=0; i<size; i++) c[i]=0; c[3]=1; c[5]=-1; linear(*this, c, t, IRT_NQ, 0);
    for (int i=0; i<size; i++) c[i]=0; c[4]=1; c[5]=-1; linear(*this, c, t, IRT_NQ, -1);
    branch(*this, t, INT_VAR_SIZE_MIN(), INT_VAL_MIN());
  }
  virtual IntVar cost(void) const { return finish; }
  virtual IntMinimizeSpace* copy(bool share) {
    return new StreamSchedule(share, *this);
  }
  StreamSchedule(bool share, StreamSchedule& s) : IntMinimizeScript(share, s) {
    t.update(*this, share, s.t); finish.update(*this, share, s.finish);
  }
...
```

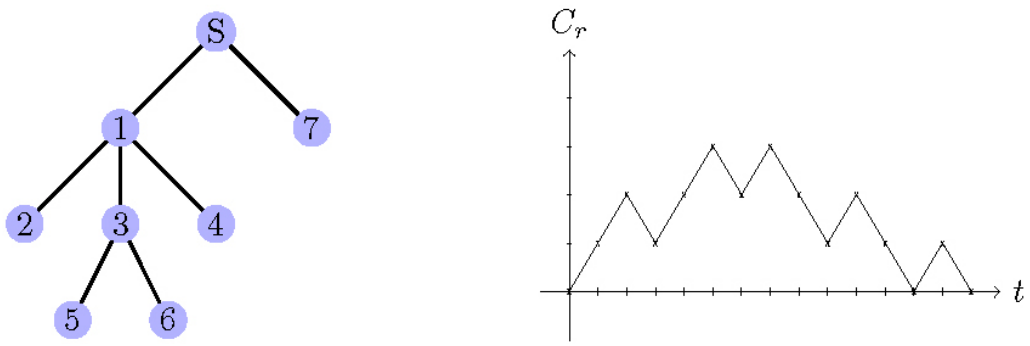
〈Figure 5〉 Example of Gecode Model

5. 계산 실험과 논의

먼저 계산 실험을 위하여 무작위로 트리들을 생성하였다. 그리고 이 트리들에 대하여 앞에서 서술한 제약식 프로그래밍 솔버인 ILOG CP와 Gecode의 입력에 필요한 모형을 생성하는 코드를 JAVA로 구현하였다. ILOG CP의 경우 OPL 언어로 구현된 모형을 직접 바이너리 솔버 프로그램에 입력을 하여 구동하면 문제에 대한 해를 산출하게 된다. Gecode의 경우는 기본적으로 C++ 언어로 표현된 솔버로, 그 입력 모형은 OPL 모형의 선언적인 표현과는 달리 C++의 클래스 같은 형태로 모형의 각 부분이 함수호출과 같은 형태로 되어 있고, 각 모형에 대해 컴파일 단계를 통해 문제를 풀어주는 바이너리 코드를 생성하도록 되어있다. 따라서 Gecode에 대해 구현된 JAVA 프로그램은 주어진 트리의 데이터를 통해 Gecode 모형을 생성하고, 생성된 모형을 풀어줄 수 있도록 해당 바이너리 코드를 생성할 수 있도록 컴파일 단계가 포함되도록 구현하였다.

일반적으로 무작위로 트리를 생성하는 방법에 대해 표준적인 방식은 정해져 있지 않아 이 논문에서는 트리의 외곽함수(contour function)을 이용하는 아이디어을 이용하였다. Le Gall[2005]에 따르면 p개의 에지를 갖고 있는 트리 τ는 외곽함수 $C_\tau(t)\,(0 \le t \le 2p)$를 통해 표현이 가능하다. <Figure 6>에서처럼 트리의 뿌리에서 시작하여 트리의 에지를 거쳐서 트리의 전역을 방문하여 다시 뿌리로 돌아온다고 할 때, 모든 에지는 노드를 방문할 때와 뿌리 쪽으로 이동할 때 두 번에 걸쳐 사용하게 된다. 이때 한 에지를 통해 이동하는데 걸리는 시간을 1로 보고, 외곽함수의 값은 한 에지를 이용하여 뿌리로부터 멀어질 때 함수의 값이 1만큼 증가하고 뿌리에 가까워지면 1만큼 감소하는 식으로 이동하여 확률과정의 랜덤 워크와 비슷한 형태를 갖는다. 따라서 외곽함수 그래프의 수평축은 방문의 시간을 나타내고 수직축은 방문한 노드에서 뿌리까지의 거리를 나타냄으로써 함수를 정의하게 된다. <Figure 10>은 외곽함수 아이디어에 대한 예를 제시하고 있다. <Figure 6(a)> 위에서의 노드 방문은 트리의 뿌리인 S에서 시작하여 S-1-2-1-3-5-3-6-3-1-4-1-S-7-S순서로 진행 된다. 노드 방문 중에 뿌리로부터의 거리가 <Figure 6(b)>에 외곽함수 그래프로 표현되었다.

위의 아이디어를 이용해 무작위로 난수를 발생시켜 랜덤 워크와 비슷한 방식으로 외곽함수를 정하면 이는 트리에서의 노드 방문을 정의하게 되고 이를 통해 대응하는 트리를 구할 수 있



(a) (b)

〈Figure 6〉 Description of Tree using Coutour Function

```
contour_generate( )
input: k - a given parameter
       U - the set of moves assigned with 'up'
       D - the set of moves assigned with 'down'
output: move
    U:=D:=∅;
    For i in 1…2p
        If |U|=|D| Then                    // when located at the root
                move[i] := 'up';
                U := U + i;
        Else
                If rand() ≤ (p-|U|)k/(p-|D|)k Then
                    move[i] = 'up';
                    U := U + i;
                Else
                    move[i] := 'down';
                    D := D + i;
                End If
        End If
    End For
End
```

〈Figure 7〉 Algorithm for Random Generation of Contour Function

게 된다. 이를 위해 <Figure 7>은 난수 발생을 통해 외곽함수를 정의하는 알고리즘을 제시하고 있다. 본 실험에서는 난수 발생을 위해 CERN에서 개발한 JAVA Jet 패키지의 Mersenne Twister 방법을 사용하였고 트리의 데이터 구조를 위해 JAVA 그래프 라이브러리인 JUNG을 사용하였다. 외곽함수의 정의는 랜덤워크와는 다르게 먼저 전체 트리를 구성하는 에지 개수의 두 배($2p$)에 해당하는 움직임이 요구되고 함수 값이 0일 경우 트리의 뿌리를 방문한 경우로 외곽함수 그래프의 x-축을 따라 움직일 때 어느 함수 값이 0이 되는 어느 한 점이 있다면 그 점까지의 함수의 상승(up)과 하강(down)의 횟수는 동일해야 한다. 그리고 $2p$에 해당하는 점에서의 함수 값은 0으로 마쳐야 한다. 함수 값이 상승함에 따라 다음 번 함수의 움직임이 하강으로 바뀔 가능성이 커지도록 하기 위해 난수 값이 비교 확률$(p-|U|)^k/(p-|D|)^k$ 이하인 경우에 움직임이 상승인 것으로 그 이상이면 하강으로 정의하였다. 전체 외곽함수에서 항상 상승과 하강의 개수는 p로 동일하므로 상승이 증가하면 앞의 비교 확률은 감소하게 되어 하강이 채택될 가능성이 늘어난다. 여기서 k는 비교 시 그 중요성을 조정하는 파라메터이다. 여기서는 k가 1.2인 경우와 1.5인 경우에 트리를 생성하였다.

난수발생을 통해서 생성된 각 외곽함수에서 JUNG 라이브커리를 이용하여 만들어진 JAVA 코드에 의해 실험을 위한 트리를 만들게 된다. 이 논문에서는 링크의 개수(p)에 따라 5개의 링크로 구성된 트리에서 14개의 링크로 구성된 트리들을 각각 5개의 인스턴스씩 생성하였다. 생성된 각 트리의 노드에서는 전송해야 할 서로 다른 패킷이 존재한다고 가정하고 앞 장에서 설명된 바와 같이 패킷 전송 스케쥴 문제를 해결하기 위한 모형을 JAVA 코드로 생성하게 된다. 여기서는 ILOG CP와 Gecode 두 개의 솔버를 고려하였으므로 각 트리별 두 가지 형태의 모형이 생성된다. 그리고 생성된 모형들은 두 솔버를 이용하여 계산 최적의 패킷 전송 스케쥴을 계산하였다. <Table 2>~<Table 5>은 그 계산 결과를 보여주고 있다. <Table 2>와 <Table 4>은 두 가지 k값의 경우 계산된 결과의 평균값을 보여주고

〈Table 2〉 Comparison of Two Solver Results for $k=1.2$(average)

#link	ILOG CP			GECODE		
	#branch	#failure	time	#node	#failure	time
5	1648	762	0.01	62	30	0.04
6	1552	654	0.01	38	17	0.12
7	5275	2486	0.03	473	234	0.13
8	427425	212675	1.21	20432	10213	0.22
9	613981	305677	2.20	25481	12737	0.42
10	19709498	9775810	67.23	729918	364955	5.67
11	193692496	95714888	653.96	6854878	3427435	35.44
12	1310219910	647493049	5002.33	41115759	20557875	193.58
13	*2691784147	*1329665124	*10580.03	584659427	292329709	2887.52
14	-	-	-	9417271978	4708635984	45489.72

〈Table 3〉 Comparison of Two Solver Results for $k=1.2$(standard deviation)

#link	ILOG CP			GECODE		
	#branch	#failure	time	#node	#failure	time
5	1634	797	0.01	75	37	0.05
6	712	378	0.00	33	17	0.16
7	5388	2700	0.03	777	389	0.14
8	282879	140587	0.72	14874	7437	0.07
9	1013581	504494	3.60	40672	20336	0.43
10	15155773	7508609	48.39	549197	274599	3.66
11	225575811	111413067	742.21	7830089	3915044	40.24
12	680717059	335874429	2637.09	20085959	10042979	110.08
13	*2258384508	*1114414273	*8659.84	903421626	451710813	4226.14
14	-	-	-	10868122950	5434061475	51743.28

있고 <Table 3>과 <Table 5>는 표준편차를 보여준다.

먼저 제안된 모형방법과 솔버로 풀 수 있는 한계를 표에서 보여주고 있다. 실험에서 링크의 수가 15 이상일 경우 제안된 두 솔버로 풀 수 없었다. 풀기를 시도한 인스턴스 중에 가장 오래 걸린 경우는 32시간 가량 걸린 경우로 링크가 14개 $k=1.2$일 경우였다. 링크의 개수가 15 이상일 경우는 40시간의 계산에서도 최적해를 구할 수 없었다. 두 솔버의 계산 결과를 비교할 때 계산 시간상 Gecode을 이용한 모형의 성능이 좋음을 보여주고 있다. 주어진 시간 안에서 더 큰 크기의 문제를 풀 수 있었다. $k=1.2$에서 ILOG CP의 경우 링크 개수 13에서 세 개의 인스턴스만을 풀 수 있었다. 두 솔버의 경우 각기 계산 완료시에 탐색트리의 변수값 할당에서 실패를 기록한 회수를 실패수로 보여주고 있다. <Table 2>와 <Table 4>에서 실패수 비교에서 Gecode 모형이 ILOG CP 모형을 이용한 풀이보다 현저히 적음을 볼 수 있다.

이는 제약식 프로그래밍의 확산법(propagation)에서 훨씬 효과적인 방법을 사용하고 있음을 보여준다. ILOG CP와 Gecode에서 같은 통계량을 보여주지 않아서 직접적인 비교는 쉽지 않

〈Table 4〉 Comparison of Two Solver Results for $k = 1.5$(average)

#link	ILOG CP			GECODE		
	#branch	#failure	time	#node	#failure	time
5	1650	748	0.01	30	13	0.14
6	1332	540	0.01	22	9	0.00
7	9275	4493	0.03	978	486	0.06
8	313834	156115	0.86	13868	6931	0.16
9	2084091	1035409	6.28	81585	40789	0.39
10	25093622	12446437	77.82	955799	477896	3.22
11	36261311	17985293	142.33	1302670	651332	6.19
12	232934683	115211628	812.48	99137765	469568878	330.06
13	95277938	47094905	345.15	3192776	1596383	16.39
14	-	-	-	9130451836	4565225913	35622.29

〈Table 5〉 Comparison of Two Solver Results for $k = 1.5$(standard deviation)

#link	ILOG CP			GECODE		
	#branch	#failure	time	#node	#failure	time
5	916	420	0.01	14	7	0.32
6	1332	540	0.01	22	9	0.00
7	9275	4493	0.03	978	486	0.06
8	313834	156115	0.86	13868	6931	0.16
9	2084091	1035409	6.28	81585	40789	0.39
10	25093622	12446437	77.82	955799	477896	3.22
11	36261311	17985293	142.33	1302670	651332	6.19
12	232934683	115211628	812.48	99137765	469568878	330.06
13	95277938	47094905	345.15	3192776	1596383	16.39
14	-	-	-	9130451836	4565225913	35622.29

지만 탐색트리의 분지(branching)가 탐색트리의 노드에서 발생하고 노드에서의 분지의 개수는 같은 수의 노드를 생성함으로 간접적으로 비교할 수 있다. 표에서 ILOG CP의 분지수와 Gecode의 탐색트리 노드수의 차이가 많이 나는 것으로 보아 역시 Gecode에서의 확산법이 더 효과적이어서 전체적인 탐색트리의 크기를 줄여주는 것으로 추론할 수 있다. 전반적으로 링크의 수 증가에 따라 계산비용은 지수 함수적으로 증가함을 볼 수 있다. 이는 본질적으로 제기된 패킷 스케쥴링 문제가 NP-hard의 성격을 지니고 있기 때문이고 표준편차를 보여주는 표에서 보듯이 인스턴스마다 계산 결과의 차이도 증가하는 것을 볼 수 있다.

6. 결론 및 한계와 향후 연구 방향

이 논문은 모바일 기기 간의 무선 메쉬 네트워크에서 패킷 스트리밍을 위한 스케쥴을 설계하는 문제를 어떻게 효과적으로 해결하는지에 대해 논의 하였다. 먼저 이전의 연구에서 제시하지 못한 문제 해결을 위하여 일반적인 모델링 방법을 제시하고 이를 제약식 프로그래밍 모델로 표현하였다. 둘째로 이를 통해 이전의 연구[Kim,

2013]에서는 단순한 휴리스틱 방법만이 제기되어 왔다. 그 이유는 문제의 구조에 대한 본격적인 탐색이 부족하여 문제를 정의할 수 있는 모형이 제대로 제시되지 못한 상태에, 해밀토니안 문제의 가능해를 찾는 단순 휴리스틱을 사례별로 임시방편적으로 적용되었다. 따라서 후속 연구를 위한 해를 탐색하는 체계적인 방법을 제시하지 못하였다. 하지만 이 논문은 문제의 구조에 대한 연구를 통해 문제의 엄밀한 모형을 제시하고 모형의 최적해를 구할 수 있는 방법으로 제약식 프로그래밍 솔버를 이용한 방법을 제시하였다. 마지막으로 두 가지 제약식 프로그래밍 솔버 ILOG CP와 Gecode를 이용하여 무작위로 생성된 문제를 계산하고 얻은 결과를 비교하여 Gecode 모형을 이용한 문제 해결에 더 효과적임을 보였다.

이 연구의 한계를 논한다면 다음과 같다. 우선 이 논문에서 제시한 방법은 실제의 문제를 다룰 정도로 아직 효과적이지 못하다. 단지 이 논문에서는 최적해를 찾을 수 있는 모델링 방법을 제시하였을 뿐 의미있는 크기의 문제를 효과적으로 계산해 내는 것은 또 다른 어려운 문제로 생각된다. 하지만 이 논문에서 제시한 제약식 프로그래밍 방법은 문제의 독특한 구조에 기반한 휴리스틱 및 근사 알고리즘 등과 결합하여 탐색트리의 크기를 줄임으로써 계산에서 향후 큰 개선을 이룰 수 있는 계산 틀이라 할 수 있다. 따라서 앞으로의 연구는 이를 개선할 근사 알고리즘의 개발이 그 주제가 될 수 있다. 또한 이 연구에서 논의된 문제의 공식적인 계산 복잡도에 대한 연구와 다른 조합 최적화 문제와의 관계에 대한 연구도 앞으로의 연구 주제가 될 수 있다.

References

[1] Akyildiz, I. F., Wang, X., and Wang, W., "Wireless Mesh Networks : A Survey", *Computer Networks*, Vol. 47, No. 4, 2005, pp. 445-487.

[2] Apt, K.R., Principles of Constraint Programming, Cambridge University Press, 2003.

[3] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M., Complexity and Approximation : Combinatorial Problems and Their Approximability Properties, Springer, Berlin, 1999.

[4] Dechter, R., Constraint Processing, Elsevier Science, 2003.

[5] Diestel, R., Graph Theory, Springer, New York, 1997.

[6] Djukic, P. and Valaee, S., "Quality-of-service provisioning for multi-service TDMA mesh networks", *Managing Traffic Performance in Converged Networks*, Springer Berlin Heidelberg, 2007, pp. 841-852.

[7] Djukic, P. and Valaee, S., "Delay Aware Link Scheduling for Multi-Hop TDMA Wireless Networks", *IEEE/ACM Transactions on Networking (TON)*, Vol. 17, No. 3, 2009, pp. 870-883.

[8] Ghazvini, M., Movahedinia, N., and Jamshidi, K., "Scheduling Algorithms in Wireless Mesh Networks : A Review", 2010 Second Pacific-Asia Conference on Circuits, Communications and System (PACCS), Vol. 1, *IEEE*, 2010, pp. 86-90.

[9] Gore, A. D. and Karandikar, A., "Link Scheduling Algorithms for Wireless Mesh Networks," *Communications Surveys and Tutorials, IEEE*, Vol. 13, No. 2, 2011, pp. 258-273.

[10] Kim, N., "A Scheduling Algorithm in the Wireless Mesh Network for Real-Time

Streaming", Master's Thesis, Department of Computer Science, Yonsei University, 2013.

[11] Le Gall, J.-F., "Random Trees and Applications", *Probability surveys*, Vol. 2, 2005, pp. 245-311.

[12] Salem, N. B. and Hubaux, J. P., "A Fair Scheduling for Wireless Mesh Networks", WiMesh, No. LCA-CONF-2005-030, 2005.

[13] Schulte, C., Tack, G., and Lagerkvist, M. Z., "Modeling and programming with Gecode", Available at http://www.gecode.org/doc-latest/MPG.pdf, (Downloaded 15 August, 2015).

[14] Shetiya, H. and Sharma, V., "Algorithms for Routing and Centralized Scheduling IEEE 802.16 Mesh Networks", Proceedings of the 1st ACM workshop on Wireless multimedia *networking and performance modeling*, ACM, 2005, pp. 147-152.

[15] Vandegriend, B., Finding Hamiltonian cycles : algorithms, graphs and performance, Master of Science, Department of Computing Science, University of Alberta, 1998.

[16] Van Hentenryck, Pascal, *Constraint Satisfaction in Logic programming*, Vol. 5, MIT Press, Cambridge, 1989.

[17] Van Hentenryck, Pascal, "Constraint programming in OPL", *Principles and Practice of Declarative Programming*, Springer, Berlin, 1999.

[18] Wei, H.-Y., Ganguly, S., Izmailov, R., and Haas, Z. J., "Interference-aware IEEE 802.16 WiMax mesh networks", In Vehicular Technology Conference, VTC 2005-Spring, Vol. 5, 2005, pp. 3102-3106.

저자소개

Hak-Jin Kim

Hak-Jin Kim is a professor of Operations Research in the School of Business at Yonsei University, Seoul, Korea. He holds a Ph.D. in Operations Research from Tepper School of Business in Carnegie Mellon University, an MS in Mathematics from University of Illinois at Urbana-Champaign, and a BBA from Yonsei University. He is currently interested in constraint programming, logic-based optimization, the integer programming, Semantic Web and AI techniques. He has published in Decision Support Systems, Annals of Operations Research, Knowledge Engineering Review, Telematics and Informatics and many other journals.