공학박사학위 논문

공항 지상 운용을 위한
선입 선처리 기반 스케줄링 기법 연구

A Study on Scheduling Algorithms
based on the First-Come First-Served Approach
for Airport Surface Operation

2020년 8월

인하대학교 대학원

항공우주공학과

박 배 선

공학박사학위 논문

공항 지상 운용을 위한
선입 선처리 기반 스케줄링 기법 연구

A Study on Scheduling Algorithms
based on the First-Come First-Served Approach
for Airport Surface Operation

2020년 8월

지도교수 이 학 태

이 논문을 박사학위 논문으로 제출함

이 논문을 박배선의 박사학위논문으로 인정함.

2020년 8월

주심 최 기 영 (인)

부심 이 학 태 (인)

위원 유 창 경 (인)

위원 전 대 근 (인)

위원 이 금 진 (인)

초 록

항공기는 오늘날 인류가 활발히 이용하는 교통수단 중 하나이다. 항공 교통량은 꾸준히 증가하고 있으며, 교통의 흐름 또한 복잡해지고 있다. 심화되는 교통 문제를 해결하기 위해선 공항과 공역 등의 인프라 확충뿐만 아니라 효율적으로 교통을 관리하기 위한 시스템이 필요하다. 현재 많은 국가에서 항공 교통 문제를 해결하기 위해 다양한 차세대 항공 교통 관리 시스템을 개발하고 있다.

스케줄링 알고리즘은 이러한 항공 교통 관리 시스템을 구성하는 중요한 요소들 중 하나이다. 현재까지 최적화 기법을 포함한 다양한 알고리즘이 연구되어 왔다. 최적화 알고리즘은 주어진 문제의 최적 해를 계산하기 때문에 최적의 스케줄링 결과를 제공할 수 있다. 그러나 가장 직관적이며 빠른 방법은 선입 선처리 알고리즘과 같이 제약 조건을 고려해 주어진 스케줄을 바로 계산하는 것이다.

본 논문에서는 공항 지상 운용을 위한 선입 선처리 기반 스케줄링 알고리즘을 제시한다. 일반적인 선입 선처리 기법에 다양한 제약 조건을 반영하고, 주어진 순서대로 스케줄링하지 않고 순서에 변화를 주는 것을 가능하도록 함으로써 그 효율과 성능을 최적화 기법에 유사한 수준으로 향상시켰다. 제시된 스케줄링 기법은 노드-링크 형태로 표현될 수 있는 모든 시스템에 적용할 수 있으며, 주어진 상황에 따라 각 항공기의 최적 출발 및 도착 시간을 계산한다. 또한, 주어진 제약 조건을 모두 만족하는 결과를 빠른 시간 내에 도출할 수 있다.

공항 지상 운용을 위한 다양한 제약 조건을 고려해 공항 스케줄링을 수행하였으며, 최적화 기반 알고리즘과의 비교를 통해 본 연구에서 제시된 기법의 성능이 최적화 기법과 유사한 수준임을 확인하였다. 최종적으로, 이를 실제 운용 환경과 연동한 수정 스케줄링 기법은 일정 주기마다 항공기 상태를 기반으로 스케줄링을 반복해 충실도 높은 스케줄링 결과를 제공할 수 있다. 본 논문에서는 실제 운용 환경을 배속 시뮬레이션으로 대체하였으며, 실제 데이터를 기반으로 인천 국제공항에서의 스케줄링을 수행하였다.

핵심어: 스케줄링, 선입 선처리 알고리즘, 노드-링크 모델, 공항, 항공 교통 관리

Abstract

Aircraft has become one of the most popular means of transportation. Air traffic is steadily increasing, and the traffic flows are becoming more complex. To solve the severe traffic problems, not only the expansion of the infrastructures such as airports and airspaces but also the systems for efficient traffic management are necessary. Currently, many countries are developing various next generation air traffic management systems.

Scheduling algorithms are one of the important elements of those air traffic management systems. Several scheduling algorithms have been studied, including optimization-based methods. An optimization algorithm calculates the optimal solution for a given problem, so it can produce an optimal scheduling solution. However, the most intuitive and efficient method is to directly calculate a given schedule satisfying the constraints, such as algorithms based on First-Come First-Served(FCFS) principle.

This dissertation proposes scheduling algorithms based on the FCFS approach for airport surface operation. A typical FCFS algorithm has been improved such that the various constraints can be considered and the flight sequence can be switched, so the scheduling performance is to that of optimization-based algorithms at a much lower computational cost. The proposed scheduling techniques can be applied to any problems that can be formulated in a node-link structure. It computes the optimal departure or arrival times of each aircraft based on a given priority and provides the result which satisfies all the given constraints almost instantly.

Airport scheduling was performed with various realistic constraints for airport ground operation, and the FCFS and the optimization-based approaches were compared. The computed schedules from FCFS showed slightly larger delays than those of an optimization-based algorithm with the computation time smaller by at

least an order of magnitude. Finally, a corrective scheduling algorithm that interacts with the real operation was developed. It repeats the scheduling process based on the aircraft states and provides scheduling results that are better suited for field application. In this dissertation, the real operation was replaced with a fast-time simulation, and the scheduling was tested at Incheon International Airport based on the historic data.

Keywords: Scheduling, First-Come First-Served(FCFS) Algorithm, Node-Link Model, Surface, Air Traffic Management

목 차

표 목차

그림 목차

1. 서론

1.1. 항공 교통 관리

1903년 최초의 동력 비행 이후, 항공기는 오늘날 인류가 활발히 이용하는 교통수단 중 하나로 자리 잡았다. 자동차 및 철도 산업이 상용화 이후 현재까지 성장해왔듯이, 항공 산업 또한 항공기의 상용화 이후 꾸준히 성장해왔다. 2017년 국제민간항공기구(International Civil Aviation Organization, ICAO) 가입국의 전체 항공운송실적은 9천 454억 톤킬로미터이며, 대한민국은 가입국 중 8번째로 많은 운송실적을 기록하였다 [1]. ICAO는 2032년까지 세계 여객 킬로미터가 연평균 4.6% 증가할 것으로 전망하였고, 국제항공운송협회(International Air Transport Association, IATA)는 2036년까지 세계 여객수가 매년 3.6%의 성장률을 보일 것으로 발표하였다 [1].

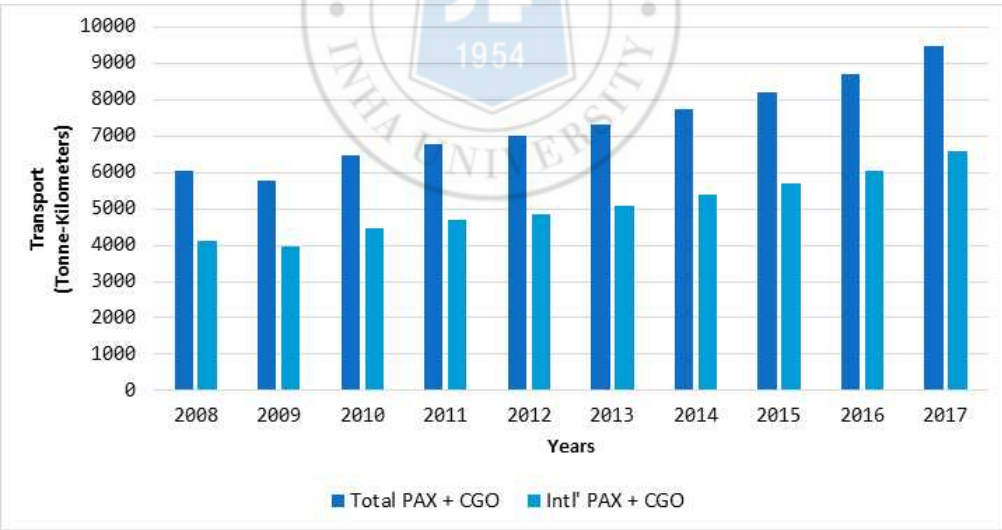


그림 1.1 ICAO 가입국 전체 항공운송실적 추이 [1]

항공 교통은 타 교통과 유사한 역할을 수행하지만, 운항 범위가 매우 크고 교통 흐름이 복잡하다. 또한 항공 교통량이 빠르게 증가하면서, 조종사가 주위를 살피기 힘들고

속도가 빠른 항공기의 특성 상 항공 사고의 발생 가능성 역시 높아지게 되었다. 이로 인해 항공기의 운항이 지상에서 엄격하게 관리될 수 있도록 하는 항공 교통 관리(Air Traffic Management, ATM) 개념과 항공 교통 관리 시스템(Air Traffic Management System, ATMS)이 등장하였다.

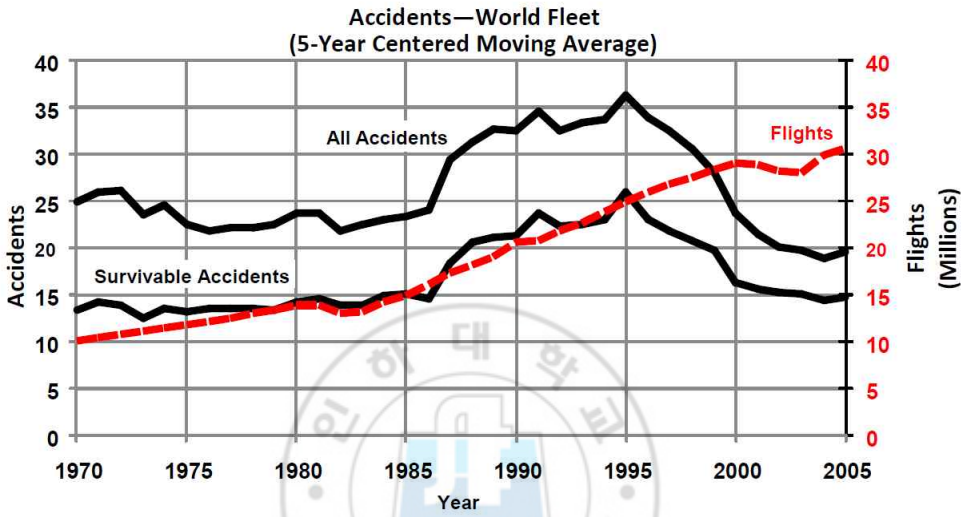


그림 1.2 연간 항공 사고 발생 수 및 항공 교통량 [2]

'항공 교통 관리'란 안전하고 경제적이며 효율적인 항공 운항을 위해 공중과 지상의 모든 집단이 협력하여 공역과 항공 교통을 역동적, 통합적으로 관리하는 것을 말한다 [3]. 항공 교통 관리 시스템은 2차 세계 대전을 전후로 미국과 유럽에서 운용되기 시작하였으며, 레이더와 자동화 기술이 도입되면서 더욱 발전하였다 [4].

1970년대 이후, 지속적으로 늘어나는 항공 교통량으로 인해 공역이 혼잡해지고 이로 인한 비행 연착 문제가 심화되기 시작하였다. 혼잡한 공역과 항공 교통을 효율적으로 관리하기 위해 항공 교통 흐름 관리(Air Traffic Flow Management, ATFM)의 중요성이 대두되었으며, 미국과 유럽을 포함한 주요 선진국들은 항공 교통 관리 시스템을 개선하기 위해 다양한 노력을 기울여 왔다. ICAO는 Aviation System Block Upgrades(ASBU), 미국과 유럽의 경우 각각 Next Generation Air Transportation System(NextGen), Single European Sky ATM Research (SESAR)와 같은 다양한 차

세대 항공 교통 관리 시스템 개발 로드맵을 수립하여 현재까지 연구 개발을 수행하고 있다 [5-8].

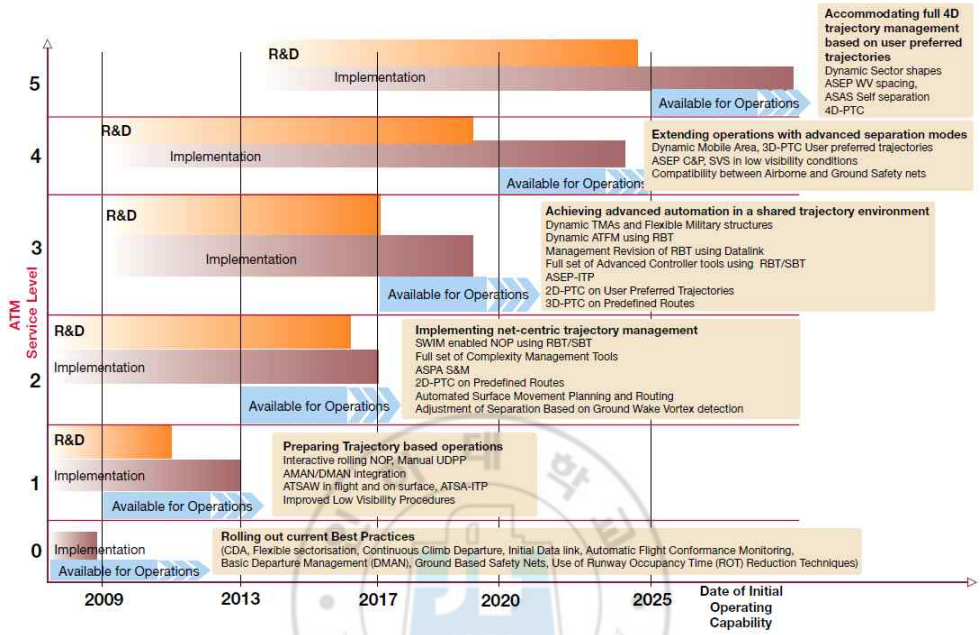


그림 1.3 SESAR R&D Master Plan [8]

1.2. 항공기 출도착 관리

공항은 항공기 운항의 출발점이자 도착점이며, 수많은 항공기들이 공항을 중심으로 모이고 흩어지기를 반복한다. 항공 교통 관리를 통해 항공기들을 공중에서 효율적으로 관리하더라도, 공항을 중심으로 병목 현상이 필연적으로 발생하게 된다. 이는 항공기 출도착 시간 지연 및 공항 처리량 감소의 원인이 되고, 나아가 공역과 항공 교통 관리의 효율성을 감소시킬 수 있다. 이는 '출발 관리(Departure management, DMAN)'와 '도착 관리(Arrival management, AMAN)' 기술을 연구하는 계기가 되었으며, 현재 유럽과 미국은 AMAN/DMAN 통합 운용 연구를 활발히 진행하고 있다 [8-13].

항공기 도착 관리는 출발 관리에 비해 먼저 연구되었으며, 도착 항공기들의 순서와 시간을 계산하여 최적화된 도착 순서를 제공한다. 특히 유럽은 1990년대 후반부터 일

부 국가에서 실제 공항에 AMAN(Arrival Manager) 시스템을 도입해 도착 관리를 수행하고 있다 [9, 10].

출발 관리 기술은 공항의 활주로 용량과 접근 관제 공역을 최적으로 사용하기 위해 항공기의 이륙 지연을 없애고 안전한 이륙 스케줄과 항공기 궤적을 제공한다 [10]. 유럽은 DMAN(Departure Manager) 시스템을 개발하여 현재 공항에서 AMAN과 DMAN을 연계하여 운용하고 있다 [8, 12, 13]. 미국은 SARDA(Spot and Runway Departure Advisor)를 개발하여 Human in-The Loop(HiTL) 시뮬레이션 등 다양한 연구를 수행하였으며, 최근 Charlotte International Airport(CLT)을 대상으로 IADS(Integrated Arrival, Departure, and Surface) 시스템의 1단계 시범 운용을 완료하였다 [14-20]. 대한민국은 현재 국토교통부 주도 하에 '항공기 출발 및 도착 통합 관리(Management on Integrated operations of Departure, Arrival, and Surface, MIDAS)' 연구가 진행되고 있으며, 향후 제주 국제공항(Jeju International Airport, CJU)과 인천공항(Incheon International Airport, ICN)을 대상으로 시스템을 구축하고 운영하는 것을 목표로 하고 있다 [10, 21, 22, 23].

스케줄링 알고리즘은 차세대 항공 교통 관리 시스템을 구성하는 중요한 요소들 중 하나이다. 스케줄러는 주어진 상황과 조건을 고려해 항공기의 순서와 시간을 계산하며, 시스템이 최선의 결과를 제공할 수 있도록 한다. 이후 1.3절에서는 다양한 스케줄링 알고리즘 연구 사례를 조사하고 분석한다.

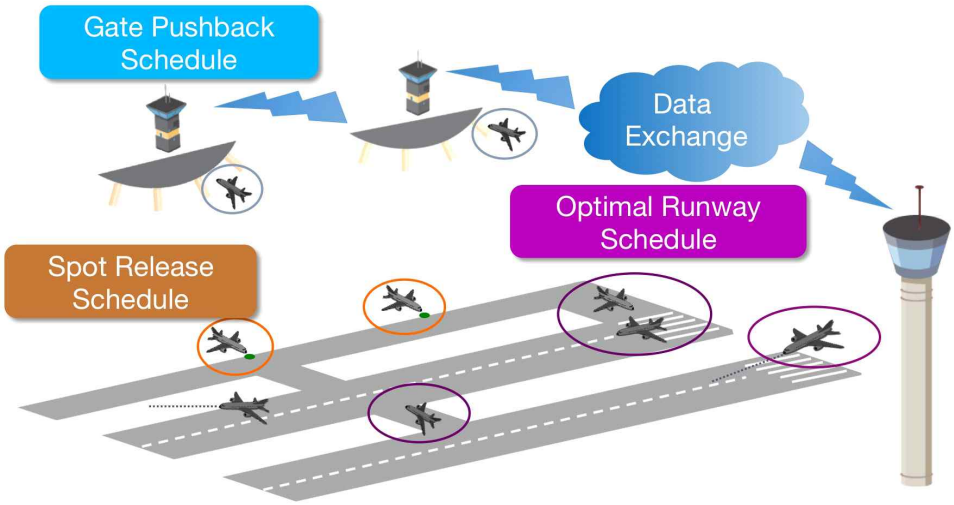


그림 1.4 SARDA Concept [14]

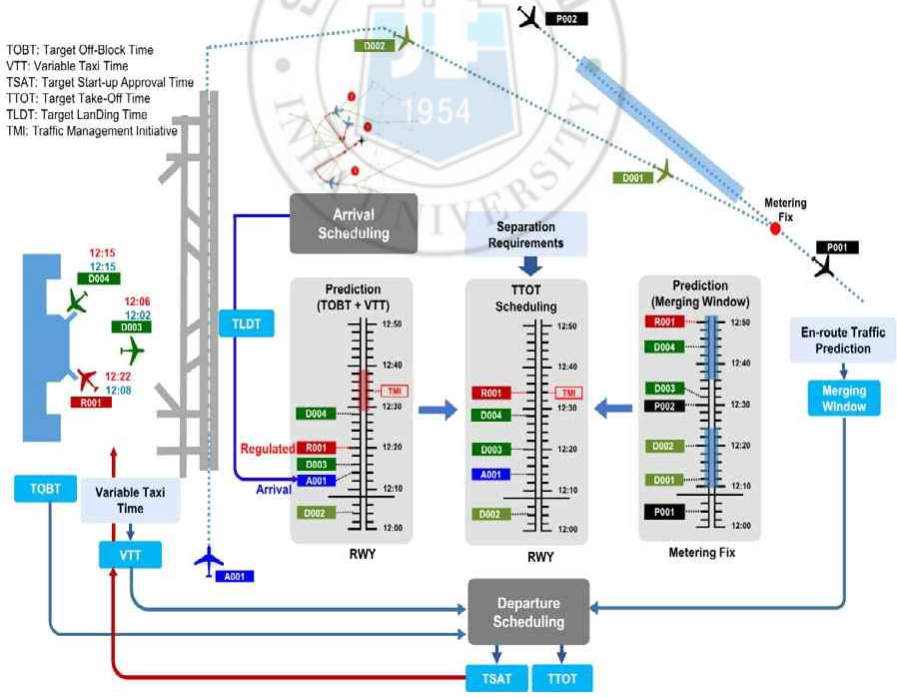


그림 1.5 MIDAS Operation Concept [21]

1.3. 유사 연구 사례

스케줄링 알고리즘은 교통 흐름 관리(Traffic Flow Management, TFM), 공항 택싱(Taxiing), 출도착 관리 등 다양한 항공 교통 문제를 해결할 수 있는 핵심 요소이다. '선입 선처리(First-Come First-Served, FCFS)' 알고리즘은 가장 일반적으로 사용되는 스케줄링 방식이다. 대부분 공항에서는 항공기들의 출발 순서를 FCFS 원칙에 따라 먼저 출발 요청을 한 항공기들을 우선적으로 배치하는 방식으로 스케줄링을 수행한다 [18]. 도착 항공기 또한 예정된 착륙 시각 순서대로 배치하여 관제사들은 항공기들의 최소 분리 간격만을 조정하도록 한다 [24]. 이러한 선착순 처리 방식은 운용자의 업무 부담을 줄이고 항공기들이 배치되는 것에 대해 형평성이 보장된다는 장점이 있으나, 공항의 항공기 처리량(Throughput)을 감소시킬 수 있다 [24]. 또한, 항공기의 출도착 시각의 순서만을 배치하고 다른 요인을 고려하지 않아 유도로와 활주로 대기열(runway queue)에서의 대기 시간을 증가시킬 수 있다 [18]. 이는 전체적인 지연 시간과 연료 소모량을 증가시키며, 결국 공항의 운용 효율을 떨어뜨리는 문제를 야기한다. 이러한 문제를 해결하기 위해 다양한 스케줄링 기법들이 연구되었으며, 대부분의 연구는 주어진 문제에 대해 최적의 결과를 제공하는 최적화 알고리즘을 기반으로 수행되었다.

일부 연구자들은 TFM 문제를 해결하기 위해 '0-1 정수 계획법(0-1 Integer Linear Programming, 0-1 ILP)' 기반 스케줄링 기법을 제시하였으며 [25, 26], '혼합 정수 선형 계획법(Mixed Integer Linear Programming, MILP)'을 활용하여 터미널 공역(Terminal airspace)에서 항공기 순서 및 경로점 진입 시간을 조정하는 최적 스케줄링 기법을 연구하였다 [27-29]. (1.1), (1.2)는 Bertsimas와 Stock-Patterson [26]이 제시한 0-1 ILP 모델이다. '목적 함수(Objective function)'는 지상과 공중에서의 항공기 지연으로 발생하는 비용을 최소화하며, 그 형태는 (1.1)과 같다. 공역의 수용량과 공항 출도착 활주로 용량이 제약 조건으로 고려되었다. (1.2)는 해당 모델의 '결정 변수(Decision variable)'로, 정의된 조건에 따라 0 또는 1의 값으로 결정된다.

$$\min \sum_{f \in F} \begin{bmatrix} \left(c_f^g - c_f^a\right) \sum_{t \in T_f^k,\, k \in P(f,1)} t\left(w_{ft}^k - w_{f,t-1}^k\right) \\ + c_f^a \sum_{t \in T_f^k,\, k \in P(f,N_f)} t\left(w_{ft}^k - w_{f,t-1}^k\right) \\ + \left(c_f^a - c_f^g\right) d_f - c_f^a r_f \end{bmatrix} \quad (1.1)$$

$$w_{ft}^j = \begin{cases} 1 & \text{if } flight\ f\ arrives\ at\ sector\ j\ by\ time\ t, \\ 0 & otherwise \end{cases} \quad (1.2)$$

일반적인 항공기 출도착 스케줄링은 '활주로 스케줄링(Runway scheduling)'과 '택시 스케줄링(Taxi scheduling)'으로 구분된다. 활주로 스케줄링을 통해 항공기의 이착륙 시간을 제공하며, 택시 스케줄링은 활주로 스케줄링 결과와 택시 예상 시간을 바탕으로 항공기의 게이트 출발 시간 및 도착 시간을 계산한다. 연구자들은 MILP를 활용하여 출도착 스케줄링의 최적 해를 구하고자 하였다 [23, 30-37]. Visser와 Roling [30]은 유도로(Taxiway) 상의 두 항공기 간 최소 분리 거리(Minimum separation distance)와 각 유도로의 항공기 수용량을 고려한 MILP 모델을 제시하였다. Smeltink 외 3인 [31]은 유도로가 교차하는 지점을 기준으로 최소 분리 거리를 적용하였고, Rolling Horizon 기법을 변형해 택시 스케줄링을 수행하였다. Rathinam 외 2인 [32]은 최소 분리 제약조건을 세분화하고, 단순화된 MILP 모델을 제시하였다. Montoya 외 3인 [33]은 다수의 이동 경로를 고려한 MILP 기반 택시 스케줄링을 통해 도착 항공기의 지상 이동 시간이 크게 감소하였음을 확인하였다.

Gupta 외 2인 [34]은 MILP를 기반으로 한 확정적 활주로 출발 스케줄링 기법을 제시하였으며, 전체적인 지연 시간의 감소가 공항의 항공기 처리량을 향상할 수 있음을 확인하였다. Malik 외 2인 [35]은 MILP를 활용하여 다수의 활주로를 대상으로 한 활주로 스케줄링 기법을 제시하였다.

Eun 외 9인 [23]은 MILP 기반의 활주로 및 택시 스케줄링 모델을 구성하여 출도착 통합 스케줄링을 수행하였으며, Clare와 Richards [36], Lee와 Balakrishnan [37]은 택시 스케줄링과 활주로 스케줄링을 동시에 고려한 MILP 기반 스케줄러를 구성하였다. (1.3-6)은 Eun 외 9인 [23]이 구성한 MILP 모델의 목적 함수와 결정 변수를 보여준다. (1.3)은 활주로 스케줄러의 목적 함수로 총 이륙 지연 시간을 최소화하며, 결정 변

수는 (1.4)와 같이 정의된다. z_{ij}는 항공기 i, j의 활주로 사용 순서로 0 또는 1의 값을 가지며, t_i는 항공기 i의 활주로 이륙 시간으로 정수 범위의 값으로 정의된다. 택시 스케줄러의 목적 함수와 결정 변수는 각각 (1.5), (1.6)과 같다. (1.5)는 최대 이륙 시간, 도착 항공기의 총 유도로 이동 시간(Taxi-in time) 및 출발 항공기의 유도로 이동 시간(Taxi-out time)을 최소화하여 최적의 게이트 출발 시간을 계산한다. (1.6)의 결정 변수 z_{ij}는 항공기 i, j의 노드 통과 순서로 0 또는 1의 값을 가진다. $t_{i,u}$는 항공기 i의 택시 경로 상 존재하는 노드 u를 통과하는 시간으로 0보다 큰 정수 값으로 정의된다.

$$\min \sum_{i \in D} \left(t_i - EarliestT_i \right) \tag{1.3}$$

$$z_{ij} \in \{0,1\} \quad \forall \quad i, j \in D \cup A \cup C$$
$$EarliestT_i \le t_i \le LatestT_i \quad \forall \quad i, j \in D \cup A \cup C \tag{1.4}$$

$$\min \left[\begin{array}{l} \alpha_p \sum_{i \in D, r \in R} \max\left(t_{i,r} - DesiredOffT_{i,r}, 0 \right) \\ + \alpha_d \left(\sum_{i \in D, r \in R} t_{i,r} - \sum_{i \in D, g \in G} t_{i,g} \right) \\ + \alpha_a \left(\sum_{i \in A, g \in G} t_{i,g} - \sum_{i \in A, r \in R} t_{i,r} \right) \end{array} \right] \tag{1.5}$$

$$z_{ij} \in \{0,1\} \quad \forall \quad i, j \in D \cup A \cup C$$
$$t_{i,u} \ge 0, \quad \forall i \in D \cup A, \quad u \in N \tag{1.6}$$

이러한 최적화 기법은 느린 계산 속도로 인해 실시간으로 활용하기에 어려움이 있으며, 주어진 문제와 제약 조건이 증가할수록 복잡도가 증가한다는 단점이 존재한다. 연구자들은 '발견법(Heuristic method)'을 활용하여 이러한 문제를 해결하고자 하였다 [38, 39]. Malik과 Jung [38]은 실시간 의사 결정 도구 적용을 위한 단일 활주로 스케줄링 기법을 제시하였으며, Eun 외 6인 [39]은 발견법을 활용해 인천 국제공항에서의 Minimum Departure Interval(MDI) 제약을 고려한 출발 스케줄링을 수행하였다.

1.4. 연구 내용 및 의의

기존 스케줄링 연구는 주로 최적화 기반의 알고리즘을 중심으로 수행되었으며, 택시 스케줄러와 활주로 스케줄러가 별개로 구성되었다. 최적화 기반 알고리즘은 주어진 조건으로부터 가장 최적의 해를 도출해낼 수 있어 실제 시스템을 구성하여 환경에 적용하기에 적합하다. 그러나 이러한 최적화 기법은 제약 조건이 추가되거나 문제의 사이즈가 커지면 계산의 복잡도와 시간이 크게 증가하기 때문에 대용량 스케줄링 문제를 해결하기에는 적합하지 않다. 또한, 경우에 따라 수렴하지 않거나 제약 조건을 모두 만족시키지 못하는 해를 도출할 수도 있다.

FCFS 알고리즘은 주어진 문제를 정해진 순서대로 처리하는 알고리즘이다. 이 기법은 최적화 기법과 비교하여 최적의 해를 도출하지는 못해 효율성이 부족하지만, 항상 모든 제약 조건을 만족하는 해를 도출한다. 또한, 계산의 부담이 적고 제약 조건의 수와 문제의 크기에 큰 영향을 받지 않기 때문에 대용량 스케줄링에 용이하다.

본 논문은 공항 지상 운용을 위한 FCFS 기반 스케줄링 알고리즘을 다룬다. 본 연구에서는 기존 FCFS 알고리즘을 개선해 스케줄러에서 주어진 순서대로만 스케줄링하지 않고 순서에 변화를 줄 수 있도록 하여 그 효율성을 증대시켰다. 또한, 다양한 제약 조건을 고려할 수 있도록 하여 그 성능을 최적화 기법과 유사한 수준으로 향상시켰다. FCFS 기반 스케줄링 알고리즘은 항공기들이 이동할 때 발생할 수 있는 간섭을 모두 고려하여 목표 지점에 도달할 때까지 아무런 지연 없이 이동할 수 있도록 한다. 따라서 게이트~활주로 이착륙에 해당하는 전체 경로를 한 번에 스케줄링할 수 있다. 이 기법은 노드-링크 구조로 표현 가능한 모든 시스템에 적용이 가능하며, 빠른 속도로 다양한 제약 조건을 모두 만족하는 해를 도출한다. 또한, 항공기 충돌 분리와 활주로 분리, 경로 할당 등 다양한 제약 조건 및 기능을 적용해 FCFS 기반 스케줄러를 개선하였다. 정량적 검증을 위해 최적화 알고리즘인 MILP 기반 스케줄러와 그 성능을 비교하였으며, 제시된 기법이 MILP와 유사한 수준의 성능을 가지고 있음을 확인하였다. 이를 실제 운용 환경과 연동한 수정 스케줄링 기법은 실제 항공기 상태를 기반으로 일정 주기마다 스케줄링을 반복하여 높은 충실도(Fidelity)의 스케줄링을 수행할 수 있다. 본 논문에서는 실제 환경을 배속 시뮬레이션으로 대체하여 그 결과를 분석하였으며, 실제 환경을 고려한 통합 스케줄링의 가능성을 확인하였다.

1.5. 본 논문의 구성

본 장은 항공 교통 관리 및 항공기 출도착 관리 시스템의 등장 배경과 개념에 대해 간단히 서술하였으며, 시스템의 핵심 구성 요소인 스케줄링 기법 연구 사례를 분석하여 본 연구 내용과 그 의의를 제시하였다.

2장에서는 FCFS 기반 스케줄링 알고리즘의 선행 연구를 살펴보고, 해당 알고리즘을 상세히 서술한다. 3장은 FCFS 기반 스케줄링 알고리즘을 공역 흐름 관리에 적용한 공역 스케줄러에 대해 설명한다. 4장에서는 FCFS 스케줄러를 공항 지상 운용에 적용하기 위해 수행한 연구 내용과 완성된 공항 스케줄러에 대해 상세히 다룬다. 또한, 실제 공항 데이터를 기반으로 스케줄링을 수행한 뒤 MILP 기반 스케줄러와 그 성능을 비교한다.

5장은 공항 스케줄러와 실제 운용 환경을 연동한 수정 스케줄링 알고리즘에 대해 설명하고, 스케줄링 결과를 분석한다. 다양한 불확실성 요소를 고려하고, 이를 통한 실제 환경과의 유사성에 대해 검토한다.

마지막으로 6장에서 본 연구를 요약하고 결론과 향후 계획을 제시하며 본 논문을 마무리한다.

2. 선입 선처리 스케줄링 알고리즘

본 장은 FCFS 스케줄링 알고리즘을 소개한다. 먼저 FCFS 스케줄링 알고리즘의 선행 연구에 대해 살펴보고, 노드-링크 개념을 활용하여 항공 교통 문제를 스케줄러에 적용 가능한 형태로 변환하는 방법을 설명한다. 이어서 본 논문에서 제시하는 FCFS 스케줄링 알고리즘의 특징과 그 전개 과정을 상세히 서술한다.

2.1. 선행 연구

FCFS 스케줄링 알고리즘은 먼저 프로세스를 요청한 대상을 우선적으로 처리하는 선착순 방식의 스케줄링 알고리즘이다. FCFS 알고리즘은 구조가 단순하고 주어진 문제를 빠른 시간 내에 해결할 수 있어 다양한 분야에서 활용되어 왔다.

앞서 언급한 바와 같이, 일반적인 FCFS 알고리즘은 선착순 원칙에 따라 항공기들의 출발 순서를 먼저 출발 요청을 한 순서대로 배치하며, 도착 항공기들의 경우 예정된 착륙 시각 순서대로 배치한다 [18, 24]. 이러한 방식은 한 지점에 항공기들의 순서만을 결정하고, 그 이후 경로에서 발생할 수 있는 요인을 고려하지 않기 때문에 그 효율이 매우 떨어진다. 연구자들은 대부분 고전적인 선착순 방식 대신 효율이 높은 최적화 기반의 스케줄링 기법을 연구하였으나, 일부 연구자들은 이러한 FCFS 기법을 개선하여 그 효율을 높이고자 하였다.

Meyn [40]은 FCFS 기법을 기반으로, 여러 점으로 표현이 가능한 스케줄링 문제를 해결할 수 있는 스케줄링 알고리즘을 제시하였다. Palopo 외 2인 [41], Lee 외 2인 [42]은 공역의 동적 특성을 링크 제약 조건으로 고려해 알고리즘을 확장하였다. Park 외 2인 [43]은 노드와 링크의 제약 조건을 모두 고려해 FCFS 스케줄링 알고리즘을 완성하였으며, 이를 활용해 공역 스케줄링을 수행하였다. 이러한 선행 연구들은 FCFS 기반 스케줄러가 항공기들을 단순히 주어진 요청 순서대로 배치하는 것에 그치지 않고, 이동 경로 등 다양한 요인을 고려해 효율적인 결과를 제공할 수 있도록 하였다.

2.2. Interval

본 연구에서 제시하는 FCFS 스케줄링 알고리즘을 구성하기 위해서는 먼저 '인터벌(Interval)'을 정의해야 한다. 인터벌은 특정 시간 범위를 의미하며, 다른 명칭으로는 '슬롯(Slot)'이라고 한다. FCFS 스케줄링 알고리즘은 인터벌 개념을 활용하여 항공기들이 해당 시간대를 사용할 수 있는지를 결정한다.

인터벌은 (2.1)과 같이 시작 시간(Beginning time, t_b)과 종료 시간(Ending time, t_e) 사이의 구간으로 정의되며, 그림 2.1은 이를 그림으로 나타낸 것이다. 이때 종료 시간은 시작 시간보다 작은 값을 가질 수 없다. 인터벌의 집합(Intervals)은 (2.2)와 같이 다수의 인터벌로 구성된다.

$$I=(t_b, t_e),\ \begin{cases} t_b = I[0] \\ t_e = I[1] \end{cases},\ t_b \le t_e \tag{2.1}$$

$$S=\{I_1, I_2, \cdots, I_N\} \tag{2.2}$$

그림 2.1 인터벌의 기본 정의

앞서 정의한 대로 인터벌은 시간의 범위이므로, 수학의 집합 연산에 기초하여 다음과 같이 3가지의 기본 연산을 정의할 수 있다. 이 연산은 두 인터벌 사이 연산뿐만 아니라 인터벌 집합 간의 연산도 수행할 수 있다.

(1) 교집합(Intersection, ∩)

인터벌 간에 중복되는 구간을 반환하며, 겹치는 구간이 존재하지 않을 시 공집합(∅)에 해당하는 'NaN 인터벌'을 반환한다. 두 인터벌의 교집합은 각 인터벌의 종료 시간과 시작 시간의 관계에 따라 결정되며, 그림 2.2는 그 예시이다. 인터벌 및 인터벌 집합의

교집합 연산을 식으로 나타내면 (2.3), (2.4), (2.5)와 같다.

$$(t_{b1}, t_{e1}) \cap (t_{b2}, t_{e2}) = \begin{cases} (t_{b2}, t_{e1}) & \text{if } t_{e1} \geq t_{b2} \\ NaN(\varnothing) & \text{if } t_{e1} < t_{b2} \end{cases} \quad (2.3)$$

$$S \cap I = \bigcup_{i=1}^{N} (I_i \cap I) \quad (2.4)$$

$$S_1 \cap S_2 = \bigcup_{j=1}^{M} (S_1 \cap I_j) = \bigcup_{j=1}^{M} \left\{ \bigcup_{i=1}^{N} (I_i \cap I_j) \right\} \quad (2.5)$$

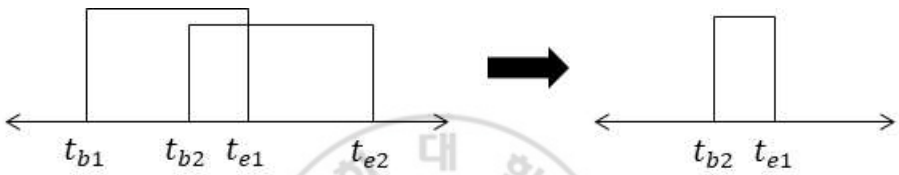


그림 2.2 두 인터벌 간의 교집합 연산

(2) 여집합(Complement, C)

대상 인터벌과 겹치지 않는 모든 구간을 반환한다. 한 인터벌의 여집합은 그림 2.3과 같이 무조건 두 개의 인터벌로 구성된 인터벌의 집합으로 반환되며, 인터벌 집합의 여집합은 각 인터벌의 여집합들의 교집합이다. (2.6), (2.7)은 인터벌과 인터벌 집합의 여집합을 식으로 표현한 것이다.

$$(t_b, t_e)^c = \{(-\infty, t_b), (t_e, \infty)\} \quad (2.6)$$

$$S^c = \bigcap_{i=1}^{N} I_i^c \quad (2.7)$$

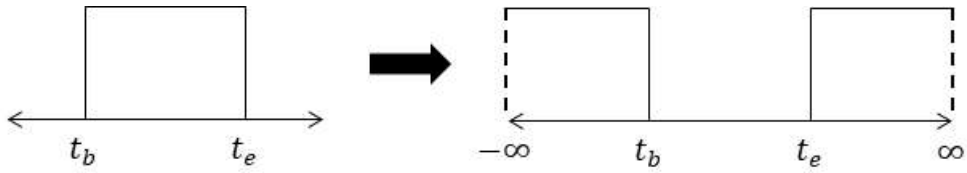


그림 2.3 인터벌의 여집합 연산

(3) 합집합(Union, ∪)

집합의 합집합 연산과 동일하며 인터벌의 집합을 반환한다. (2.8)은 두 인터벌 간의 합집합 연산을 식으로 표현한 것이다. 두 인터벌의 교집합이 존재하지 않을 경우 그 합집합은 두 인터벌로 구성되며, 교집합이 존재할 경우 합집합은 두 인터벌이 합쳐진 하나의 인터벌로 구성된다. 그림 2.4는 교집합이 존재하는 두 인터벌 간의 합집합을 보여준다. 인터벌 집합과 인터벌 간의 합집합 연산은 (2.9)와 같으며, 인터벌 집합 간의 연산은 따로 정의하지 않는다.

$$(t_{b1}, t_{e1}) \cup (t_{b2}, t_{e2}) = \begin{cases} \{(t_{b1}, t_{e2})\} & \text{if } t_{e1} \geq t_{b2} \\ \{(t_{b1}, t_{e1}), (t_{b2}, t_{e2})\} & \text{if } t_{e1} < t_{b2} \end{cases} \quad (2.8)$$

$$S \cap I = \bigcap_{i=1}^{N} (I_i \cup I) \quad (2.9)$$

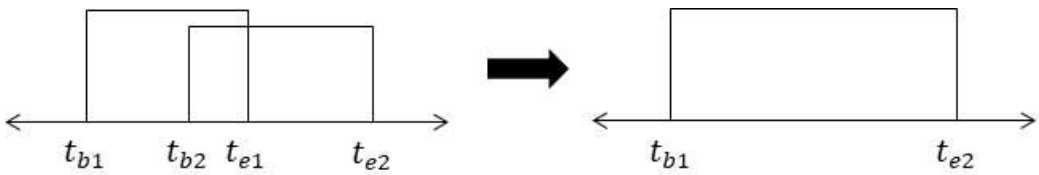


그림 2.4 두 인터벌 간의 합집합 연산

수학적 정의의 인터벌은 '열림(Open)'또는 '닫힘(Closed)'이라는 속성을 가지고 있다[44]. 열린 인터벌(Open interval)과 닫힌 인터벌(Closed interval)의 구간은 동일하지만, 시작과 끝 값의 포함 여부로 구분할 수 있으며 각각 (2.10), (2.11)과 같이 표기한다. 그림 2.5와 같이 열린 인터벌은 시작 시간과 종료 시간을 포함하지 않으며, 닫힌 인

터벌은 시작 시간과 종료 시간을 모두 구간에 포함하는 것으로 정의된다.

$$I_{open} = (t_b, t_e),\ t_b < t < t_e \tag{2.10}$$

$$I_{closed} = [t_b, t_e],\ t_b \le t \le t_e \tag{2.11}$$

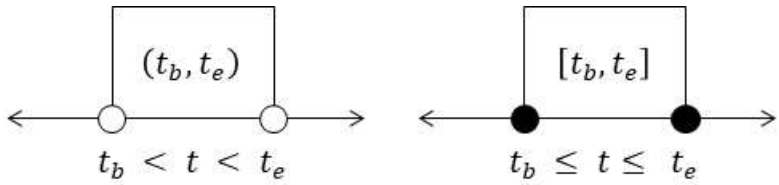


그림 2.5 열린 인터벌(왼쪽)과 닫힌 인터벌(오른쪽)

앞서 정의한 인터벌 연산에 위 개념을 적용하면 동일한 속성을 지닌 인터벌 간의 연산으로 세분화할 수 있다. 그림 2.6은 인터벌 연산을 수행할 경우 발생할 수 있는 특수한 상황을 보여준다. 이처럼 한 인터벌의 종료 시간(t_{e1})과 다른 인터벌의 시작 시간(t_{b2})이 동일한 경우($t_{e1} = t_{b2}$), 인터벌의 속성에 따라 연산의 결과가 달라질 수 있다.

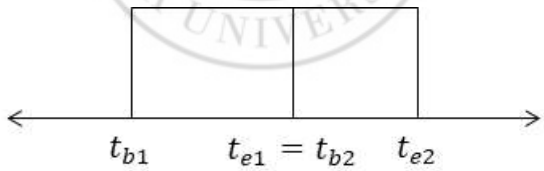


그림 2.6 인터벌 연산의 특수한 상황

(1) 교집합 → 열린/닫힌 교집합(Open/Closed intersection)

그림 2.7과 같이 두 인터벌이 모두 열린 인터벌인 경우 교집합은 존재하지 않는다. 반면 두 인터벌이 닫힌 인터벌일 경우, 교집합은 그림 2.8과 같이 시작 시간과 종료 시간이 동일한 인터벌로 존재한다. (2.12), (2.13)은 각각 열린 인터벌과 닫힌 인터벌의 교집합 연산을 식으로 나타낸 것이다.

$$(t_{b1}, t_{e1}) \cap (t_{b2}, t_{e2}) = NaN \tag{2.12}$$

$$[t_{b1}, t_{e1}] \cap [t_{b2}, t_{e2}] = [t_{e1}, t_{b2}] \tag{2.13}$$

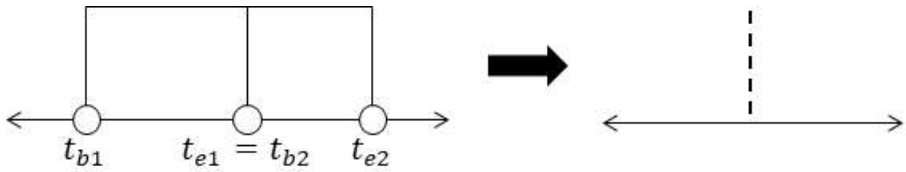


그림 2.7 열린 교집합 연산

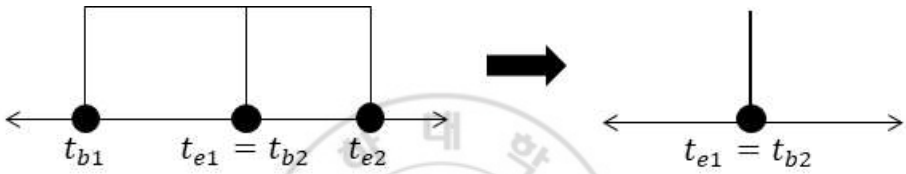


그림 2.8 닫힌 교집합 연산

(2) 합집합 → 열린/닫힌 합집합(Open/Closed union)

그림 2.9와 같이 두 인터벌이 모두 열린 인터벌인 경우, 교집합이 존재하지 않으므로 합집합은 두 인터벌로 구성된 인터벌의 집합이다. 반면 두 인터벌이 모두 닫힌 인터벌일 경우, 교집합이 존재하므로 합집합은 그림 2.10과 같이 한 인터벌의 시작 시간부터 다른 인터벌의 종료 시간까지 모두 포함하는 하나의 인터벌로 구성된다. (2.14), (2.15)는 각각 열린 인터벌과 닫힌 인터벌의 합집합 연산을 식으로 표현한 것이다.

$$(t_{b1}, t_{e1}) \cup (t_{b2}, t_{e2}) = \{(t_{b1}, t_{e1}), (t_{b2}, t_{e2})\} \tag{2.14}$$

$$[t_{b1}, t_{e1}] \cup [t_{b2}, t_{e2}] = \{[t_{b1}, t_{e2}]\} \tag{2.15}$$

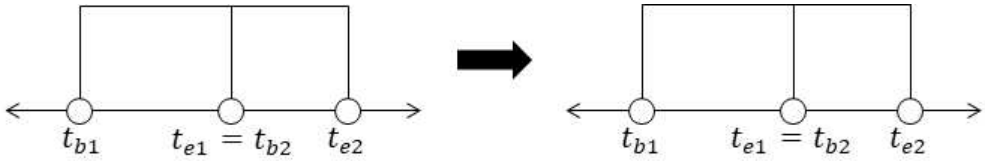


그림 2.9 열린 합집합 연산

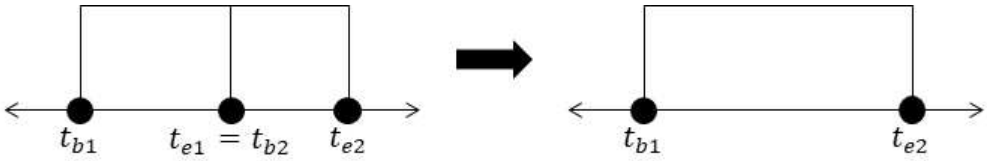


그림 2.10 닫힌 합집합 연산

인터벌을 FCFS 스케줄링 알고리즘에 적용하기 위해서는 집합 연산 외에 인터벌의 덧셈(+)과 뺄셈(−)의 정의가 필요하다. 두 연산은 인터벌 또는 인터벌 집합을 대상으로 하며, 그 정의는 (2.16−19)와 같다 [45].

$$(t_{b1}, t_{e1}) + (t_{b2}, t_{e2}) = (t_{b1} + t_{b2}, t_{e1} + t_{e2}) \quad (2.16)$$

$$\{I_1, I_2, \cdots, I_N\} + (t_b, t_e) = \{(t_{b1} + t_b, t_{e1} + t_e), \cdots, (t_{bN} + t_b, t_{eN} + t_e)\} \quad (2.17)$$

$$(t_{b1}, t_{e1}) - (t_{b2}, t_{e2}) = (t_{b1} - t_{e2}, t_{e1} + t_{b2}) \quad (2.18)$$

$$\{I_1, I_2, \cdots, I_N\} - (t_b, t_e) = \{(t_{b1} - t_e, t_{e1} - t_b), \cdots, (t_{bN} - t_e, t_{eN} - t_b)\} \quad (2.19)$$

2.3. 노드-링크 모델

‘노드(Node)’와 ‘링크(Link)’는 ‘그래프(Graph)’를 구성하는 기본 단위이며, 링크는 두 개의 노드로 이루어져 있다 [46]. 그림 2.11은 노드와 링크로 구성된 그래프이다.

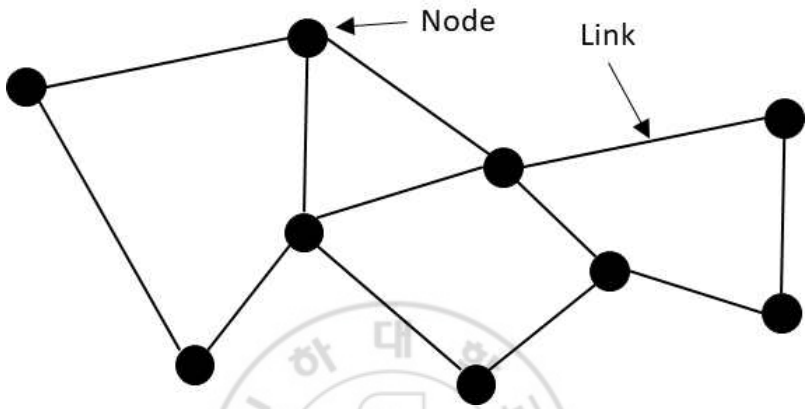


그림 2.11 노드와 링크로 구성된 그래프

일반적인 항공기의 비행경로는 ‘출발 공항 – 출발 터미널 공역 – 비행 공역 – 도착 터미널 공역 – 도착 공항’과 같이 일련의 공역(Airspace)을 통과하는 것으로 표현될 수 있다. 즉, 그림 2.12에서 공항 ‘①’에서 공항 ‘②’로 이동하는 항공기의 비행경로는 ‘① - A - B - C - D - G - ②’와 같다. 여기에 그래프 이론을 적용하면, 각 공역 하나의 링크와 같고 공역 사이의 경계는 각 링크가 연결된 노드와 같다. 따라서 항공기 비행경로는 그림 2.13과 같이 노드-링크 구조로 표현될 수 있다. 공항은 그림 2.14와 같이 공항 자체가 하나의 노드-링크 모델에 대응되므로, 공항 내에서 움직이는 항공기 경로는 자연스럽게 노드-링크 구조로 치환된다.

항공기는 노드를 순간적으로 통과하기 때문에, 노드의 제약 조건은 ‘단위 시간 동안 노드를 통과하는 항공기 수’, 즉 ‘노드 통과율(Node rate)’로 적용된다. 링크의 경우, 항공기가 링크를 이동하는 시간은 유한한 값을 가진다. 따라서 링크는 ‘현재 주어진 링크 상에 있는 항공기 수’를 제약 조건으로 고려한다.

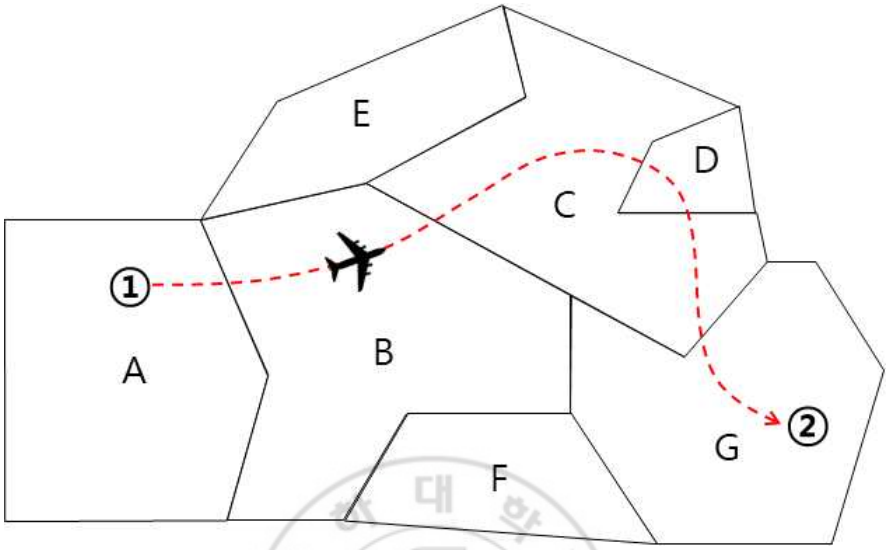


그림 2.12 일반적인 공역 상의 항공기 비행경로

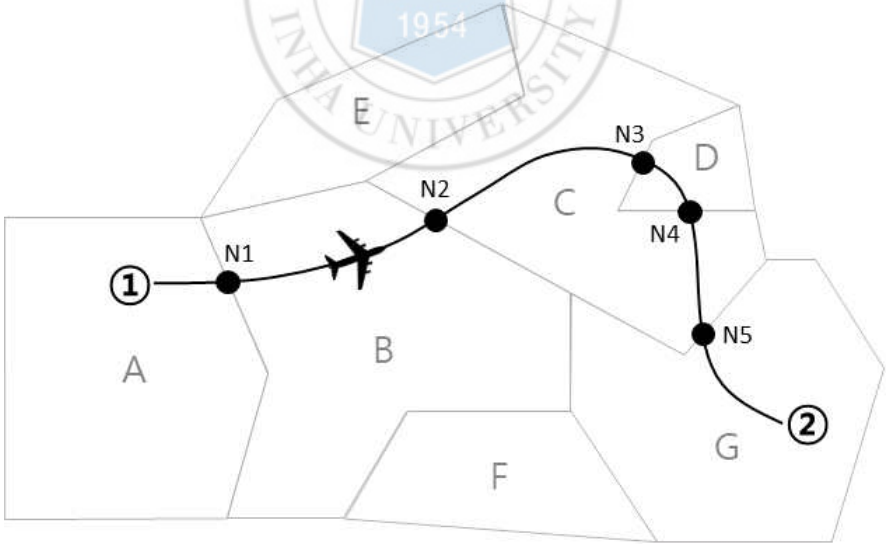


그림 2.13 그래프로 표현된 항공기 비행경로

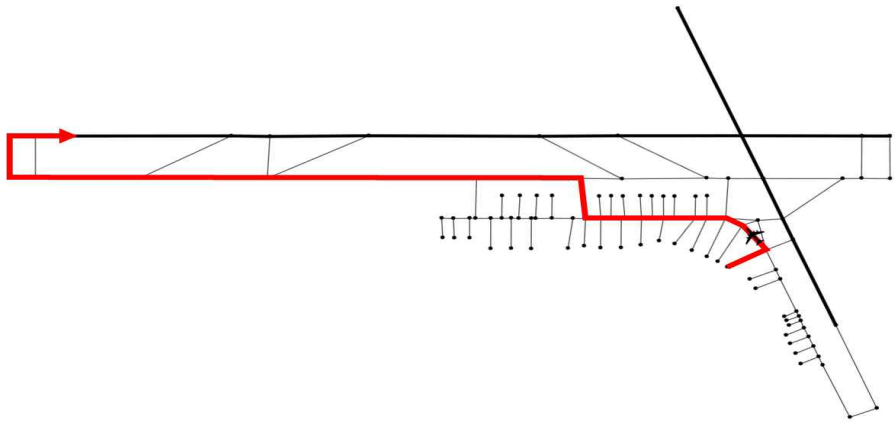


그림 2.14 제주 국제공항의 노드-링크 모델과 항공기 이동 경로

항공기 스케줄은 이동 경로 및 경로 상에 존재하는 각 링크에 진입하는 시간(Entry time)과 통과하는 시간(Exit time), 링크를 이동하는 데 걸리는 시간(Transit time) 등으로 구성된다. 그림 2.15는 그림 2.12, 2.13의 항공기 스케줄을 보여준다. 일반적으로 시간은 당일 0시를 기준으로 하며, 필요한 경우 항공기 속도 변화와 같은 조건이 추가되기도 한다.

Flight Id	Entry Time	Exit Time	Transit Time	Upper Stream Sector	Current Sector	Down Stream Sector
FLT001	1280	1650	370	-	①	A
FLT001	1650	2280	630	A	B	C
FLT001	2280	2980	700	B	C	D
FLT001	2980	3125	145	C	D	C
FLT001	3125	3585	460	D	C	G
FLT001	3585	4035	450	G	②	-

그림 2.15 항공기 비행 스케줄 예시(단위: 초)

노드-링크 구조로 표현된 항공기의 경로와 주어진 항공기의 스케줄은 그림 2.16과 같이 시간-거리 평면상에 각 노드의 타임라인을 나열하는 방식으로 나타낼 수 있다. 여기서 각 노드와 노드 사이의 공간은 링크의 타임라인이다. 항공기의 속도 변화를 고려할 경우, 가는 점선으로 표시된 범위 내에서 항공기 스케줄을 조절할 수 있다.

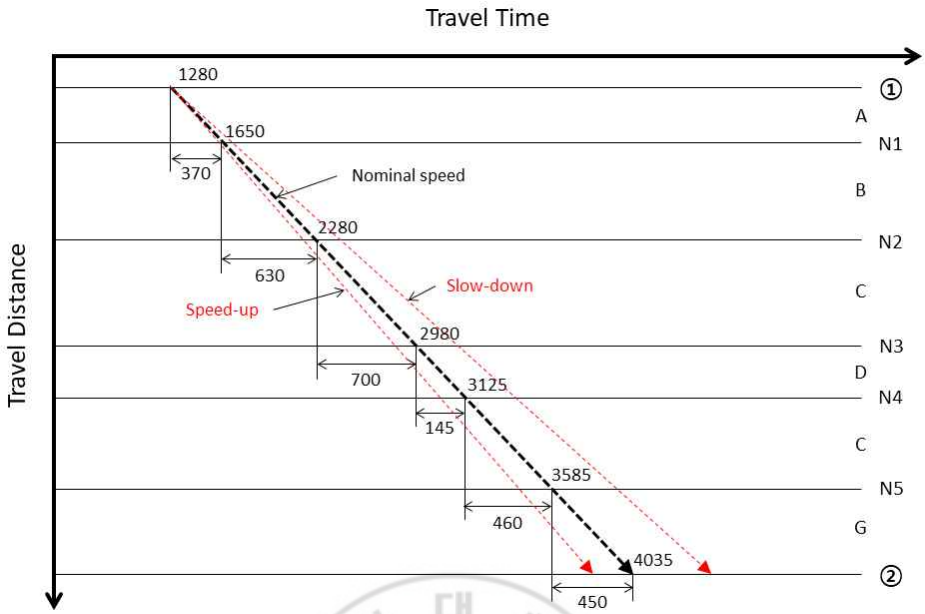


그림 2.16 시간-거리 평면에 나타낸 항공기 스케줄과 노드-링크 경로

2.4. FCFS 스케줄링 알고리즘

FCFS 스케줄링 알고리즘은 앞 절에서 설명한 노드-링크 모델의 정보와 제약 조건, 그리고 항공기 스케줄을 입력받아 스케줄링을 수행하며, 다양한 특징을 가진다.

FCFS 스케줄링 알고리즘은 '우선순위(Priority)'에 기반을 둔 순차적 스케줄링 기법으로, 높은 우선순위의 항공기가 먼저 스케줄링 되어 그 결과가 다음 순서의 항공기에 영향을 미치게 된다. 따라서 원래 비행 계획상 항공기들의 출도착 순서가 고정되지 않고 변경될 수 있으며, 이를 통해 FCFS 스케줄링 알고리즘의 효율성이 증대된다. 또한, 주어진 문제에서 전체적인 최적 해를 도출하는 것이 아닌, 정해진 순서대로 각 항공기의 최적 출발 및 도착 시간을 계산한다.

스케줄링 우선순위는 목적에 따라 다양하게 설정될 수 있다. 예를 들어 주어진 비행 계획상 항공기 출도착 시간 순서대로 스케줄링을 수행하거나, 도착 항공기들의 지연을 최소화하는 것이 목적일 경우 공항에 도착하는 항공기들을 먼저 스케줄링할 수 있다.

2.4.1. Forward Propagation

Forward propagation은 대상 항공기의 초기 스케줄을 기반으로 출발 노드에서부터 도착 노드까지 인터벌 연산을 수행하여 목적지에 가장 빨리 도착하는 시간(Earliest arrival time, t_{EA})을 계산하는 과정이다.

그림 2.17은 한 노드에 항공기들이 정해진 순서대로 배정된 것을 보여준다. FLT001의 경로가 'A - AB - BC - CD - D'와 같다면, 그 스케줄은 곧 그림 2.18과 같이 나타낼 수 있다. 이동하는 경로에 아무런 제약이 없을 경우, 항공기는 정해진 시간인 t_A에 출발하여 정해진 시간인 t_D에 무사히 도착할 것이다. 혹은 최대한 빨리 이동하여 t_D'에 도착하거나, 최대한 천천히 이동하여 t_D''에 도착할 수도 있다.

그림 2.17 항공기들이 배정된 노드 타임라인

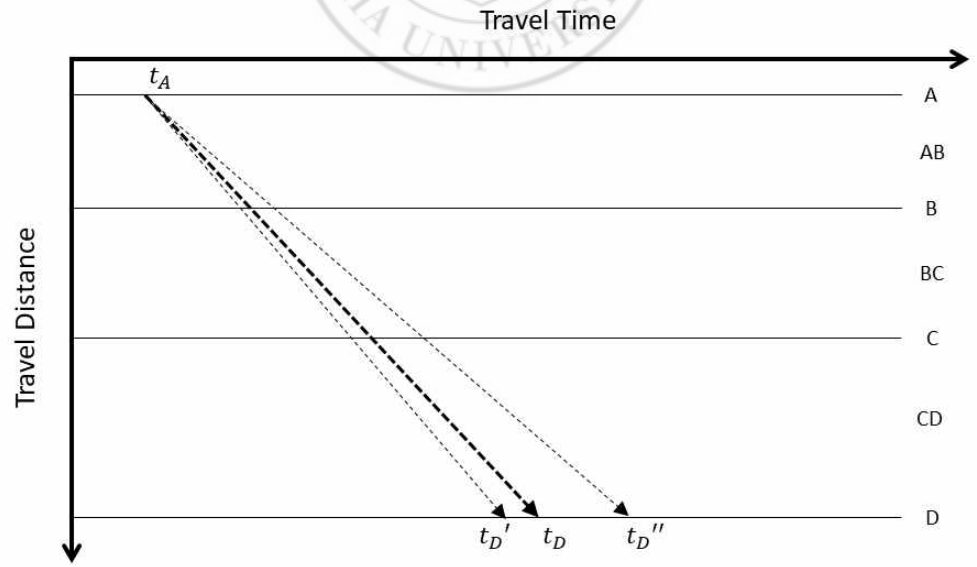


그림 2.18 항공기 FLT001의 초기 스케줄

항공기는 시간 t_A에 노드 A를 출발해 링크 AB를 이동할 수 있다. (2.20)은 항공기가 링크 AB에 진입할 수 있는 구간인 'Entry slots'를 인터벌로 정의한 것으로, 그림 2.19는 이를 그림으로 나타낸 것이다. 여기서 $AB_{ENT,f}$는 forward propagation 과정에서의 링크 AB의 entry slots, $A_{ENT,f}$는 노드 A의 entry slots를 의미한다. 이때 링크 AB에 진입하는 것은 노드 A에 진입해 순간적으로 통과하는 것과 동일하기 때문에, $AB_{ENT,f}$는 $A_{ENT,f}$와 같다.

$$AB_{ENT,f} = A_{ENT,f} = \{[t_A, \infty]\} \tag{2.20}$$

그림 2.19 노드 A에서의 entry slots

항공기들이 특정 시간에 노드 A를 통과하지 못하는 제약이 있을 수 있다. 이러한 노드 제약 조건(Node constraints)은 (2.21)과 같이 인터벌로 정의할 수 있으며, 이를 노드의 'Unavailable slots'이라 한다. (2.21)의 A_{UA}는 노드 A의 unavailable slots를 의미한다. 여기서 항공기들이 아무 제약 없이 노드를 통과할 수 있는 구간인 'Available slots'는 unavailable slots의 여집합과 같다. (2.22)는 이를 식으로 표현한 것으로, A_{AVL}은 노드 A의 available slots를 의미한다. 따라서 항공기가 노드를 실제로 통과할 수 있는 'Available entry slots'는 (2.23)과 같이 노드의 entry slots와 available slots의 교집합 연산을 통해 구할 수 있다. (2.23)의 $A_{AE,f}$는 $A_{ENT,f}$와 A_{AVL}의 교집합으로, 노드 A의 available entry slots를 의미한다. 그림 2.20은 (2.21−23)을 그림으로 나타낸 것이다.

$$A_{UA} = \{[t_1, t_2], [t_3, t_4], [t_5, t_6]\} \tag{2.21}$$

$$A_{AVL} = A_{UA}^c = \{[0, t_1], [t_2, t_3], [t_4, t_5], [t_6, \infty]\} \tag{2.22}$$

$$A_{AE,f} = A_{ENT,f} \cap A_{AVL} = A_{ENT,f} \cap A_{UA}^c \tag{2.23}$$

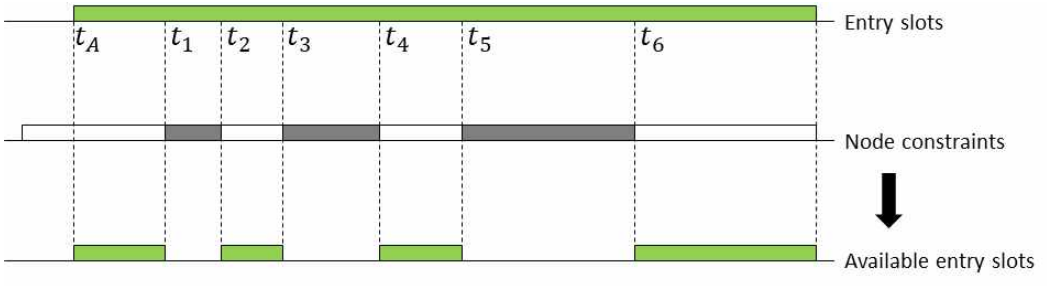


그림 2.20 노드의 available entry slots

해당 노드의 available entry slots $A_{AE,f}$가 결정되면, 그림 2.21과 같이 링크 이동 시간을 더해 다음 노드까지 전개하여 링크를 통과하는 구간인 'Exit slots'를 계산한다.

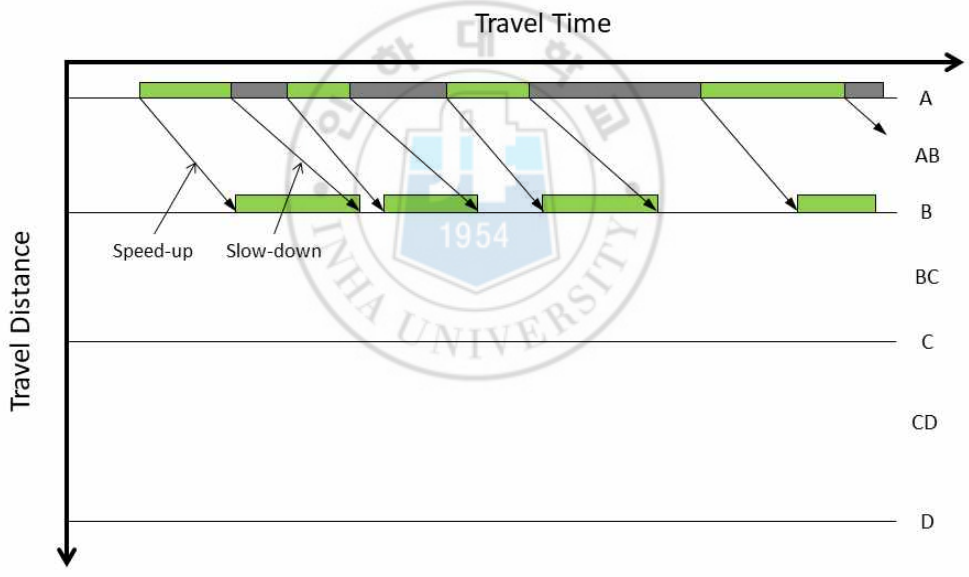


그림 2.21 노드 A의 available entry slots에서 노드 B로 전개한 모습

(2.24)는 이를 인터벌 연산으로 표현한 것으로, $AB_{EXT,f}$는 링크 AB의 exit slots, T_{AB}는 링크 AB의 이동 시간(Transit times)을 의미한다. 즉, $AB_{EXT,f}$는 $A_{AE,f}$의 각 entry slot의 시작 시간에 최소 이동 시간을 더하고, 종료 시간에 최대 이동 시간을 더하는 것으로 정의된다. 이를 통해 항공기는 다음 노드에서 최대한 넓은 entry slots를

할당받아 보다 유연한 스케줄링이 가능하다. 이때 (2.25)와 같이 $AB_{EXT,f}$은 노드 B의 entry slots인 $B_{ENT,f}$와 동일하다.

$$AB_{EXT,f} = A_{AE,f} + T_{AB} = A_{AE,f} + \{[t_{min}, t_{max}]\} \quad (2.24)$$

$$AB_{EXT,f} = B_{ENT,f} \quad (2.25)$$

노드 B에서도 노드 A와 마찬가지로 (2.23)을 이용해 available entry slots를 구한 뒤 다음 노드 C를 향해 전개한다. 그림 2.22는 이를 그림으로 나타낸 것이다.

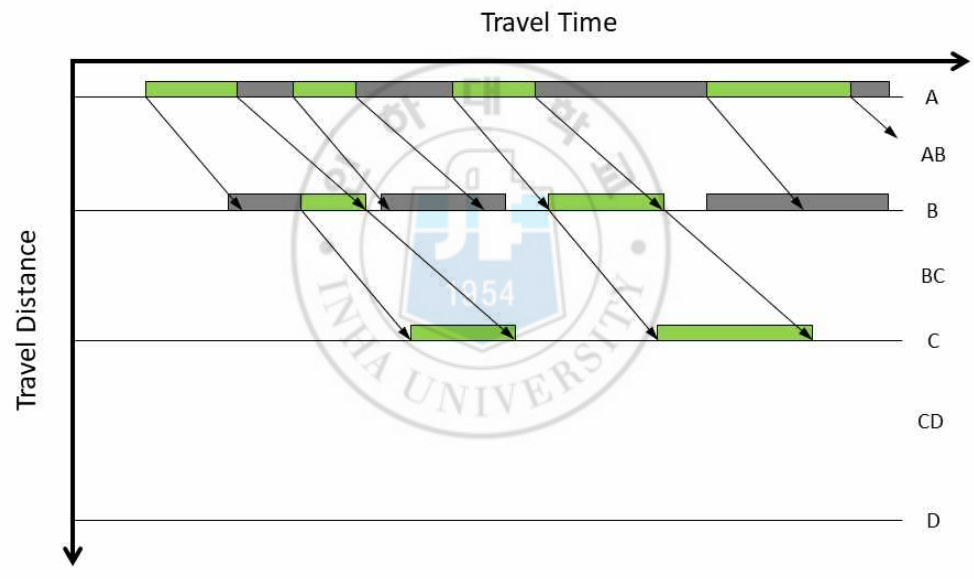


그림 2.22 노드 B의 available entry slots에서 노드 C로 전개한 모습

이때 그림 2.23과 같이 링크 BC에 제약 조건이 존재할 경우, 노드 B와 C의 available slots는 새로 계산되어야 한다. 링크 제약 조건 역시 인터벌의 집합으로 구성되며, 그 적용 방식은 노드와 유사하다. 현재 노드에서 다음 노드로 전개할 때, 링크 제약 조건은 그림 2.24와 같이 두 노드 상에 위치한 제약 조건들로 표현할 수 있다.

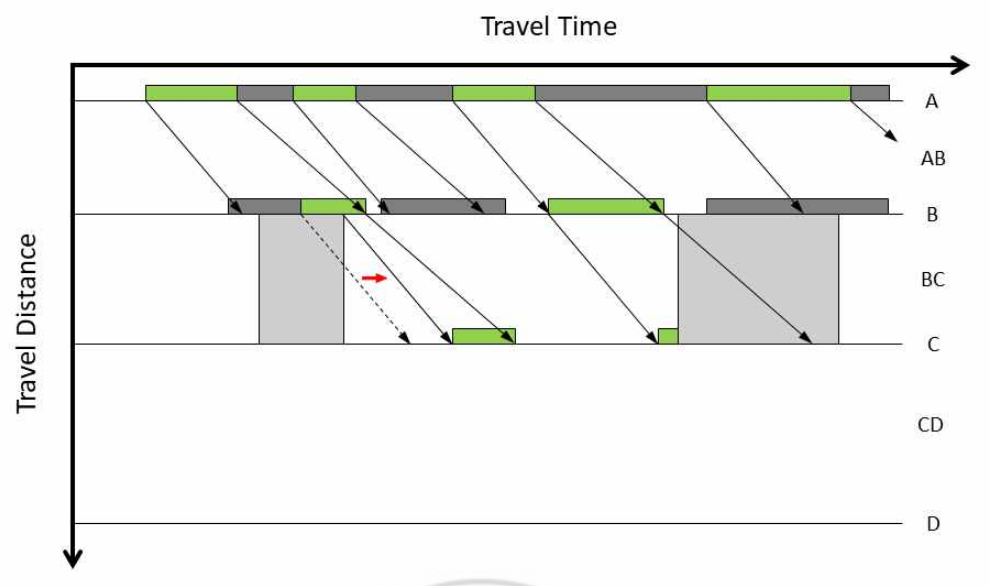


그림 2.23 링크 BC에 제약 조건이 적용된 모습

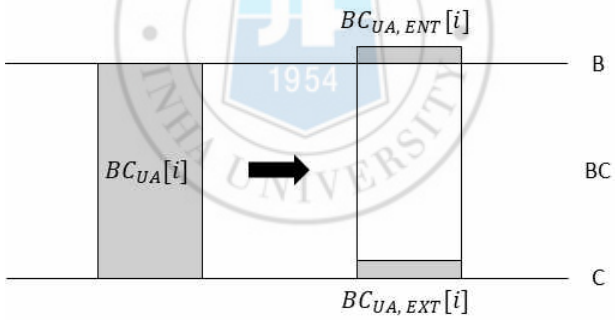


그림 2.24 노드 제약 조건들로 표현된 링크 BC의 제약 조건

진입 노드(Entry node)에 위치한 제약 조건은 링크의 'Unavailable entry slots' 또는 'Entry blocks', 통과 노드(Exit node)에 위치한 제약 조건은 링크의 'Unavailable exit slots' 또는 'Exit blocks'로 정의되며, 각 노드의 unavailable slots와 동일한 역할을 수행한다. 따라서 노드 B의 available entry slots와 링크 BC의 entry blocks를 이용해 노드 B의 available entry slots를 다시 계산해야 한다. 새로 계산된 노드 B의 available entry slots에 링크 BC에 이동 시간을 더해 노드 C의 entry slots를 구한다. 이때 노드

C의 entry slots 역시 링크 BC의 exit blocks로 인해 그 결과가 다시 계산된다. 이러한 링크 제약 조건의 처리 과정은 (2.26-28)과 같이 나타낼 수 있다.

$$B_{AE,f} = B_{AE,f} \cap BC^{c}_{UA,ENT} = (B_{ENT,f} \cap B^{c}_{UA}) \cap BC^{c}_{UA,ENT} \tag{2.26}$$

$$C_{ENT,f} = B_{AE,f} + t_{BC} \tag{2.27}$$

$$C_{ENT,f} = C_{ENT,f} \cap BC^{c}_{UA,EXT} \tag{2.28}$$

이때 (2.28)에서 링크 제약 조건 $BC_{UA,ENT}[i]$가 진입 노드의 available entry slot $B_{AE,f}[j]$의 오른쪽에 위치할 경우 (2.29)와 같이 처리한다. 그림 2.25는 이를 그림으로 나타낸 것이다.

$$C_{ENT,f}[k] = \left(C_{ENT,f}[k] \cap BC_{UA,EXT}[i]^{c}\right)[0], \quad \text{if } BC_{UA,ENT}[i] \geq B_{AE,f}[j] \tag{2.29}$$

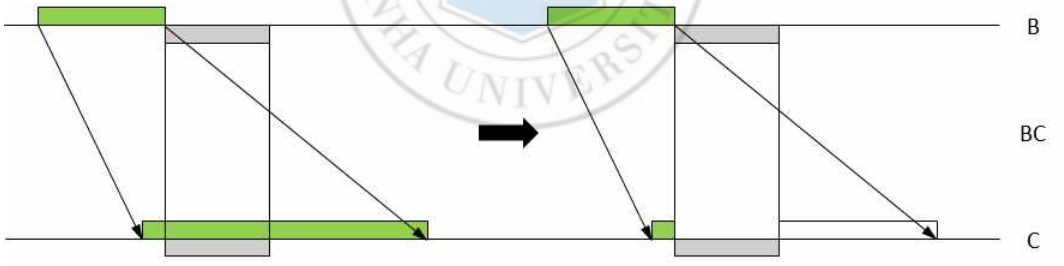


그림 2.25 링크 제약 조건이 available entry slot의 오른쪽에 위치하는 경우

이후 동일한 과정을 거쳐 스케줄을 전개해 도착 노드 D의 available entry slots를 구한다. 이때 그림 2.27과 같이 도착 노드의 available entry slots에서 가장 빠른 시간이 항공기의 earliest arrival time으로 결정되며, (2.30)은 이를 식으로 나타낸 것이다.

$$t_{EA} = D_{AE,f}[0][0] \tag{2.30}$$

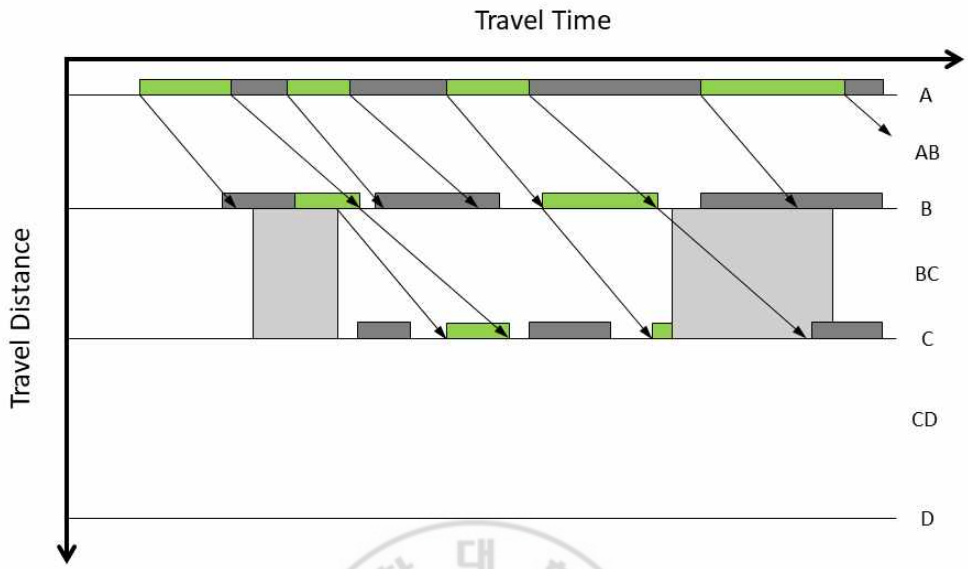


그림 2.26 노드 C의 제약 조건을 적용해 결정된 available slots

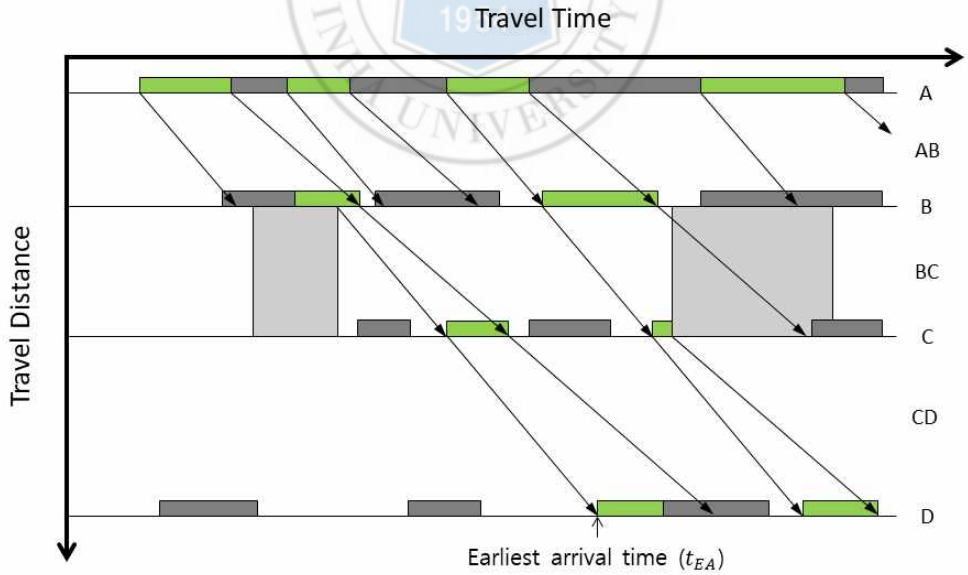


그림 2.27 노드 D의 available entry slots와 earliest arrival time

2.4.2. Backward Propagation

Forward propagation으로 t_{EA}이 결정된 후, 도착 노드에서부터 출발 노드까지 backward propagation을 수행해 항공기가 가장 빨리 출발할 수 있는 시간(Earliest departure time, t_{ED})을 결정한다.

그림 2.28의 녹색 영역은 앞서 수행한 forward propagation을 통해 결정된 노드 A부터 D까지 항공기가 이동할 수 있는 영역이며, backward propagation은 도착 노드인 D를 시작점으로 수행된다. 이때 노드 D는 forward propagation의 도착점이자 backward propagation의 시작점이므로 (2.31)에서와 같이 $D_{AE,b}$는 $D_{AE,f}$와 동일하며, 이는 항공기가 아무 제약 없이 노드 D를 사용할 수 있음을 의미한다.

$$D_{AE,b} = D_{AE,f} = D_{AE} \tag{2.31}$$

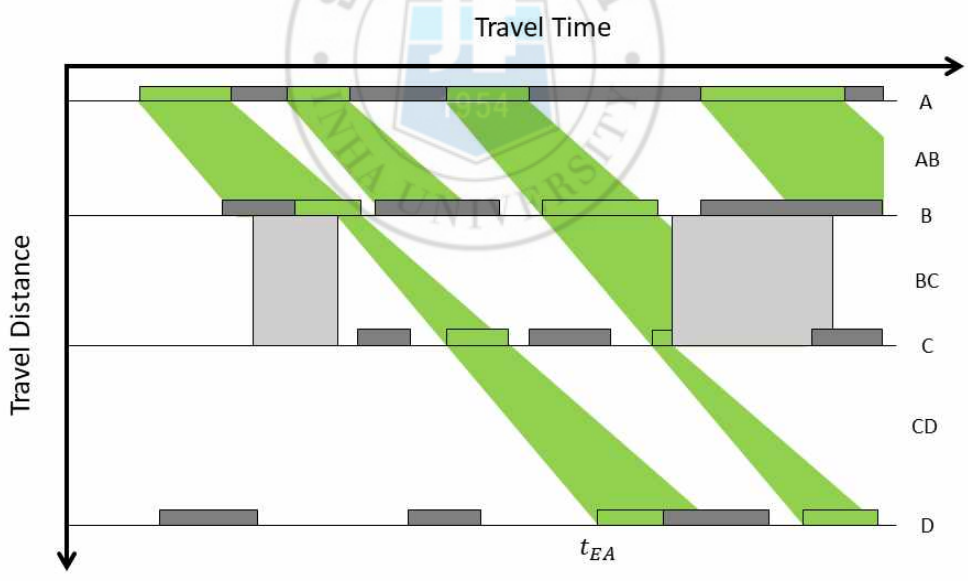


그림 2.28 Forward propagation을 통해 결정된 항공기가 이동 가능한 영역

먼저 (2.32)와 같이 도착 노드 D의 첫 번째 슬롯 $D_{AE,b}[0]$에서 노드 C까지 역 전개

를 수행해 노드 C에서의 entry slot을 구한다. $C_{ENT,b}$는 backward propagation 과정에서 계산된 노드 C의 entry slot, T_{CD}는 링크 CD에서의 이동 시간을 의미한다. 즉, $C_{ENT,b}$는 $D_{AE,b}[0]$의 시작 시간에서 최대 이동 시간을 빼고, 종료 시간에서 최소 이동 시간을 빼는 것으로 정의된다.

따라서 노드 C의 역방향 available entry slots $C_{AE,b}$는 $C_{ENT,b}$와 forward propagation을 통해 구한 $C_{AE,f}$의 교집합이다. (2.33)은 이를 식으로 나타낸 것이다. 그림 2.29는 노드 D에서 노드 C까지의 backward propagation을 그림으로 나타낸 것이다. 이때 항공기가 아무 제약 없이 노드 C를 통과할 수 있는 구간 C_{AE}는 $C_{AE,b}$와 동일하다.

$$C_{ENT,b} = D_{AE,b}[0] - T_{CD} = D_{AE,b}[0] - \{[t_{min}, t_{max}]\} \quad (2.32)$$

$$C_{AE,b} = C_{AE,f} \cap C_{ENT,b} = C_{AE} \quad (2.33)$$

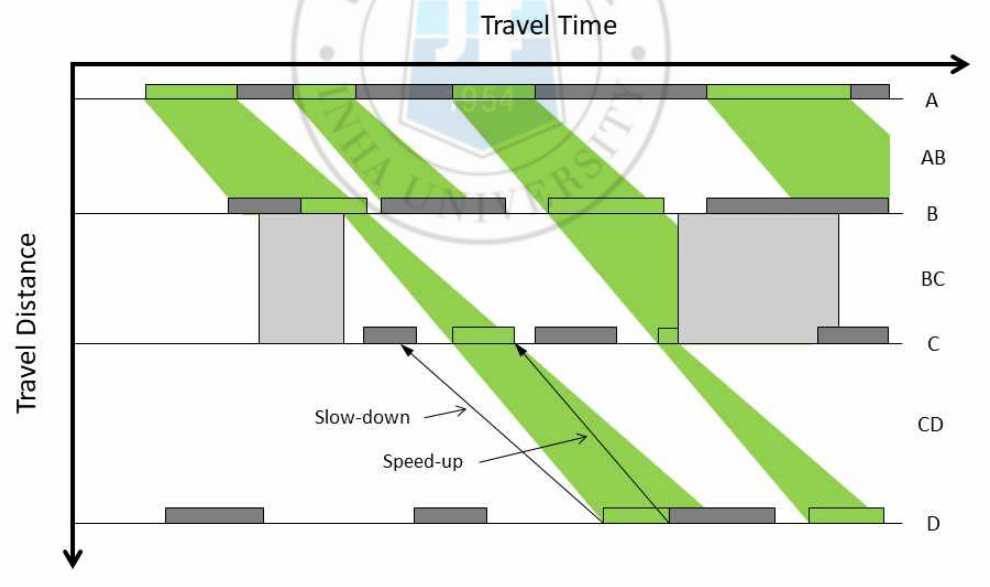


그림 2.29 노드 D에서 노드 C까지 역 전개한 모습

노드 C에서 노드 B까지의 과정도 이전과 동일하게 노드 C의 available entry slots의

첫 번째 슬롯 $C_{AE}[0]$을 기준으로 수행해 노드 B의 available entry slots를 구한다. 링크 BC에 제약 조건이 존재하므로 역방향 이동 시에도 제약이 없도록 해야 한다. Backward propagation에서의 링크 제약 조건 처리 과정은 (2.34)와 같으며, 이때 링크의 entry blocks, exit blocks는 전개 방향과 관계없이 그림 2.24와 동일하게 적용한다.

$$B_{AE} = B_{AE,b} \cap BC^{c}_{UA,ENT} = \left(B_{AE,f} \cap B_{ENT,b}\right) \cap BC^{c}_{UA,ENT} \tag{2.34}$$

여기서 링크 제약 조건 $BC_{UA,EXT}[i]$가 역진입 노드의 available entry slot $C_{AE}[j]$의 오른쪽에 위치할 경우 (2.35)와 같이 처리한다. 그림 2.30은 이를 그림으로 나타낸 것이다.

$$B_{AE}[k] = \left(B_{AE,b}[k] \cap BC_{UA,ENT}[i]^{c}\right)[1], \quad \text{if } BC_{UA,EXT}[i] \geq C_{AE}[j] \tag{2.35}$$

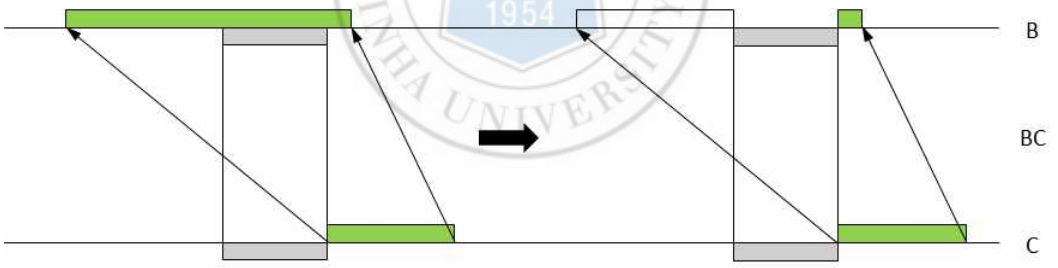


그림 2.30 링크 제약 조건이 available entry slot의 왼쪽에 위치하는 경우

위 과정을 시작 노드 A까지 반복해 backward propagation을 완료하면 그림 2.31과 같이 earliest departure time이 결정된다. Earliest departure time과 earliest arrival time이 결정되면 자연히 항공기의 스케줄 및 시작점에서의 지연 시간(Delay)이 결정되며, 이는 그림 2.31과 같다. 그림 2.32에서 녹색 영역과 파란색 영역이 겹치는 영역은 항공기가 노드 A에서 노드 D 사이들 아무 제약 없이 이동할 수 있는 영역이며, 굵은 실선은 항공기가 가장 빠른 속도로 이동하는 경로를 보여준다.

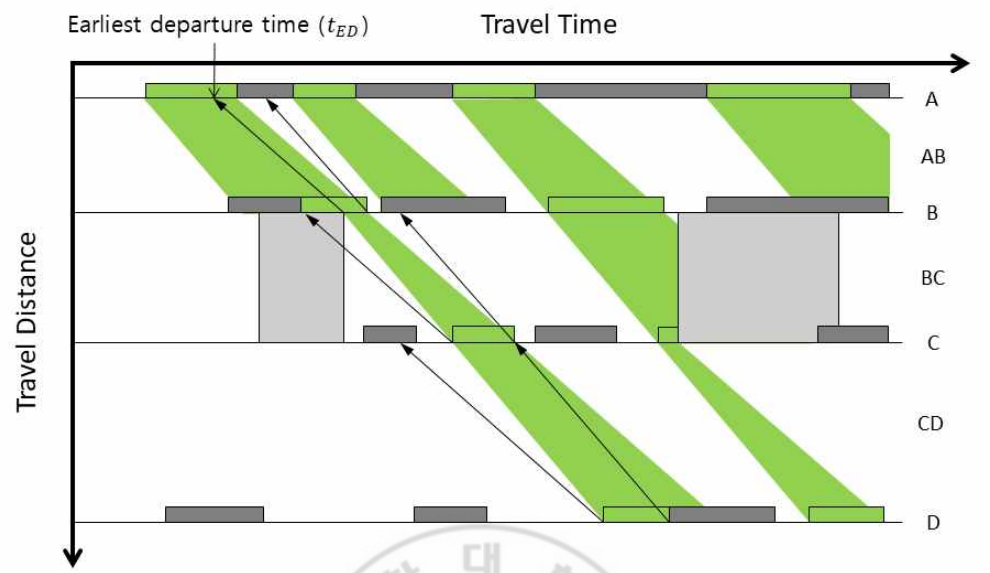


그림 2.31 Backward propagation을 통해 결정된 노드 A의 earliest departure time

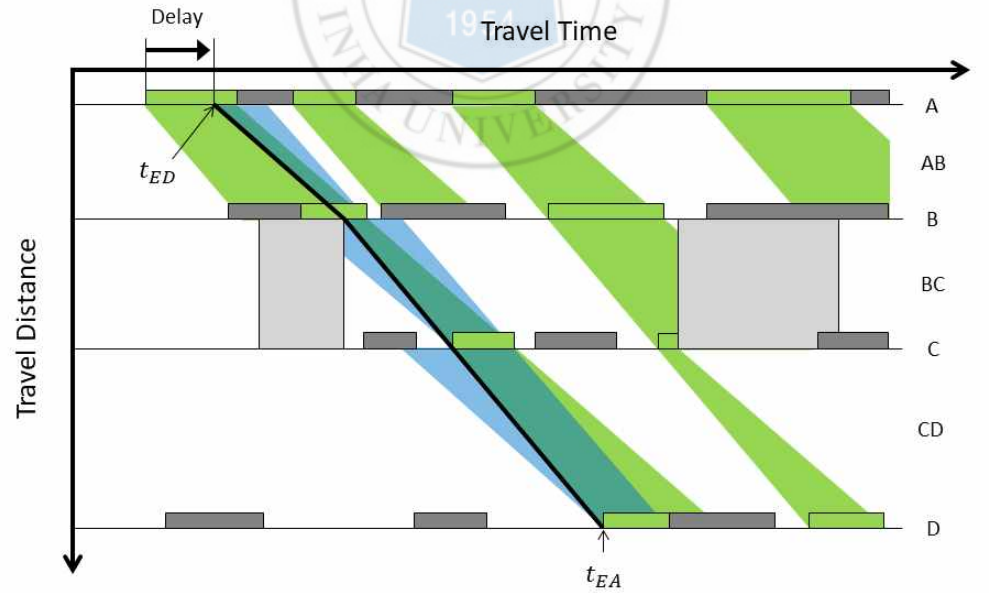


그림 2.32 최종적으로 결정된 항공기의 이동 경로와 지연 시간

한 항공기에 대해 모든 propagation이 완료되고 항공기의 스케줄이 새롭게 정해지면 다음 항공기의 스케줄링을 위해 노드-링크 모델을 업데이트한다. 이때 업데이트되는 노드와 링크는 그림 2.33의 빗금 처리된 영역과 같이 스케줄링이 완료된 항공기의 이동 경로이다. 각 노드와 링크의 업데이트 과정은 스케줄링 알고리즘의 적용 분야에 따라 다르므로, 해당 과정은 이어지는 3장 및 4장에서 상세히 설명한다.

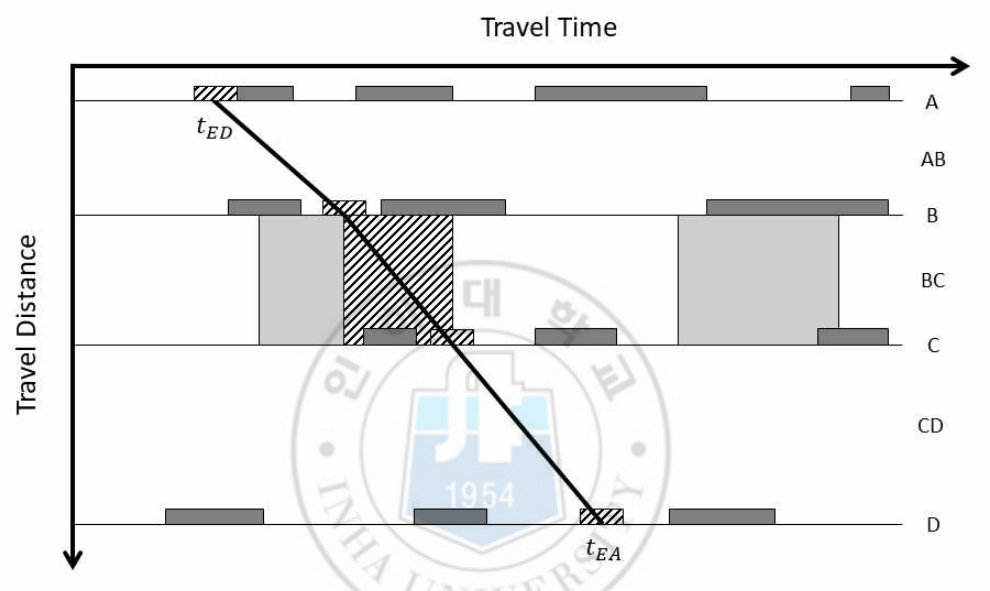


그림 2.33 스케줄링 완료 이후 업데이트된 노드-링크

3. 공역 스케줄링

본 장은 2장에서 설명한 FCFS 스케줄링 알고리즘을 적용한 공역 스케줄링에 대해 다룬다. 공역에서의 노드-링크 모델의 특징을 설명하고, FCFS 스케줄링 알고리즘이 공역의 노드-링크 제약 조건을 어떠한 형태로 적용하는지에 대해 상세히 설명한다.

3.1. 공역 노드-링크 아키텍처

공역 스케줄링에서의 노드는 공항과 각 공역 간의 경계, 링크는 항공기가 지나는 공역에 대응된다. 그림 3.1은 다수의 항공기가 공역을 통과하는 예시이다. 이때 노드는 항공기들이 경계를 통과하는 지점이기 때문에 특정 위치로 한정할 수 없다. 따라서 노드 제약 조건은 공역 경계를 제외한 출도착 공항, 즉 시작 노드와 도착 노드에서만 고려한다. 링크의 경우 항공기들이 공역에 자유롭게 진입하고 통과하는 것이 가능하므로, 한 링크에 대해 물리적 제약을 정하는 것이 불가능하다. 따라서 링크는 공역의 항공기 수용량을 제약 조건으로 고려한다.

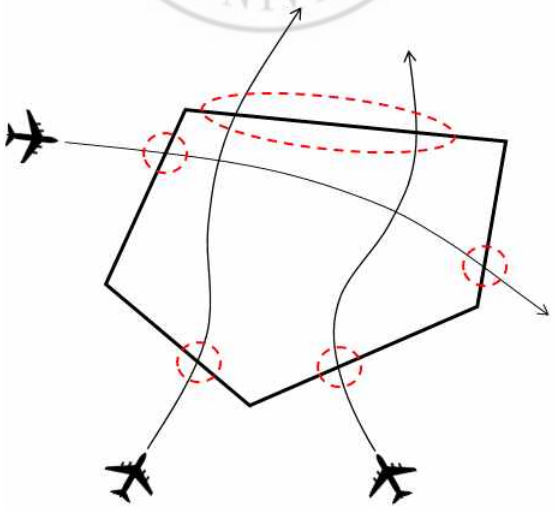


그림 3.1 공역을 통과하는 항공기들

항공기 1대의 스케줄링 이후 출도착 노드의 업데이트는 해당 항공기의 출도착 시간(t_{ED}, t_{EA})을 기준으로 unavailable slot을 하나 추가하는 것이다. (3.1)은 시간 t를 기준으로 한 노드 N의 unavailable slot N_{UA}를 식으로 표현한 것이다. 인터벌 N_{UA}는 $2\Delta t$의 크기를 가지며, 이를 그림으로 나타내면 그림 3.2와 같다.

이때 노드 제약 조건의 크기를 결정하는 Δt는 노드 통과율에 의해 결정된다. 공역 스케줄링에서 출도착 노드의 통과율은 공항의 'Aircraft Departure Rate(ADR)' 및 'Aircraft Arrival Rate(AAR)'을 사용한다. ADR과 AAR은 각각 1시간 동안 공항에서 출발하고 도착하는 항공기의 수를 의미한다. 따라서 Δt는 (3.2)와 같이 ADR, AAR의 단위 시간인 1시간을 해당 시간대의 ADR과 AAR로 나눈 것과 같다.

이와 같이 항공기 1대의 스케줄링을 완료한 뒤 출도착 노드를 업데이트하면 해당 구간은 새로운 노드 제약 조건으로 적용되어 다른 항공기들이 통과할 수 없다. 즉 노드 제약 조건은 최대한 지연이 없도록 스케줄링을 수행하더라도 설정된 노드 통과율을 절대 초과하지 않도록 한다.

$$N_{UA} = [t-\Delta t, t+\Delta t],\ t = t_{ED} \text{ or } t_{EA} \tag{3.1}$$

$$\Delta t = \frac{1\ hour}{ADR \text{ or } AAR} \tag{3.2}$$

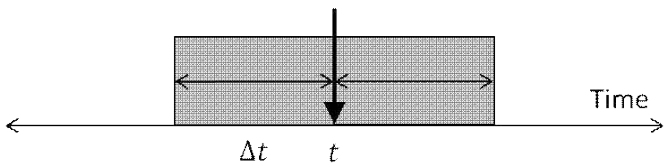


그림 3.2 출도착 노드 제약 조건

'단위 시간당 항공기 통과 횟수'를 제약 조건으로 적용하는 공항과 달리 공역은 '공역 내 항공기 수(Count)'를 제약 조건으로 적용해 available slot과 unavailable slot을 결정한다. 공역의 가장 큰 특징은 그림 3.3과 같이 한 공역이 수용할 수 있는 항공기의 수가 정해져 있고, 단일 공역에 다수의 항공기가 동일한 시간에 진입하거나 통과할 수 있

다는 것이다. 공역 내 항공기 수가 해당 공역의 수용량과 같은 경우 그 공역은 포화 상태로 간주되어 공역의 항공기 수가 감소할 때까지 다른 항공기들은 해당 공역에 진입할 수 없다. 따라서 링크의 제약 조건은 해당 공역에서의 항공기 수가 공역 수용량과 동일한 구간이 unavailable slots로 적용된다.

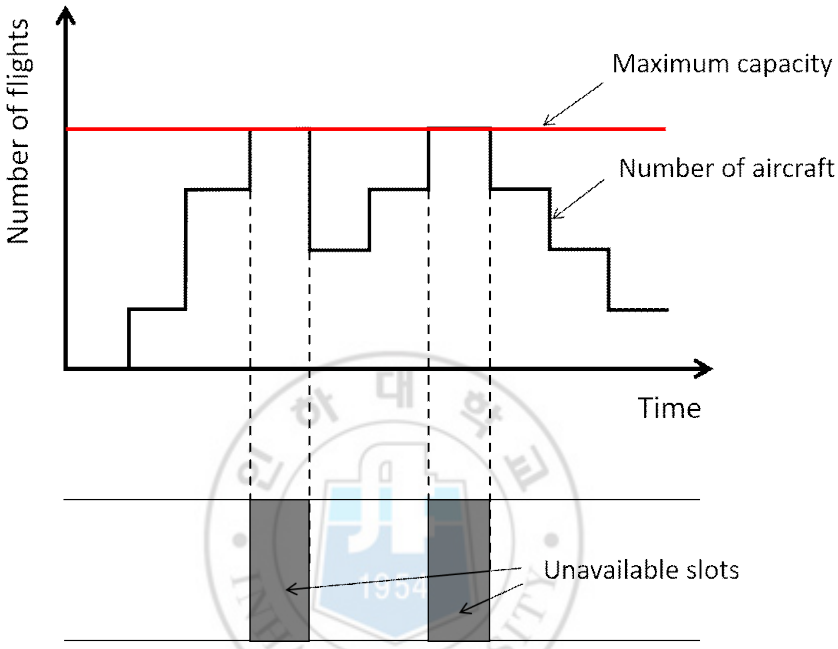


그림 3.3 공역 링크 아키텍처

3.2. 스케줄링 결과 [47]

2015년 4월 1일부터 4월 2일까지 약 48시간의 대한민국 전체 항적 데이터를 기반으로 공역 스케줄링을 수행하였다. 항적 데이터의 끝점이 대부분 인천 비행정보구역(Flight Information Region, FIR) 외부에 위치하기 때문에, 인천 FIR을 둘러싼 4개의 가상 공역이 존재한다고 가정하였다. 인천 FIR 외부, 즉 가상 공역에 위치하는 국제선 항적의 시작점과 끝점은 각각 출발 공항과 도착 공항으로 가정하였다. 또한, 가상 공역과 가상 공항은 항공기들이 아무 제약 없이 이동할 수 있다고 가정하였다. 그림 3.4는 인천 FIR을 둘러싼 가상 공역 및 항적 데이터의 시작점과 끝점을 보여준다.

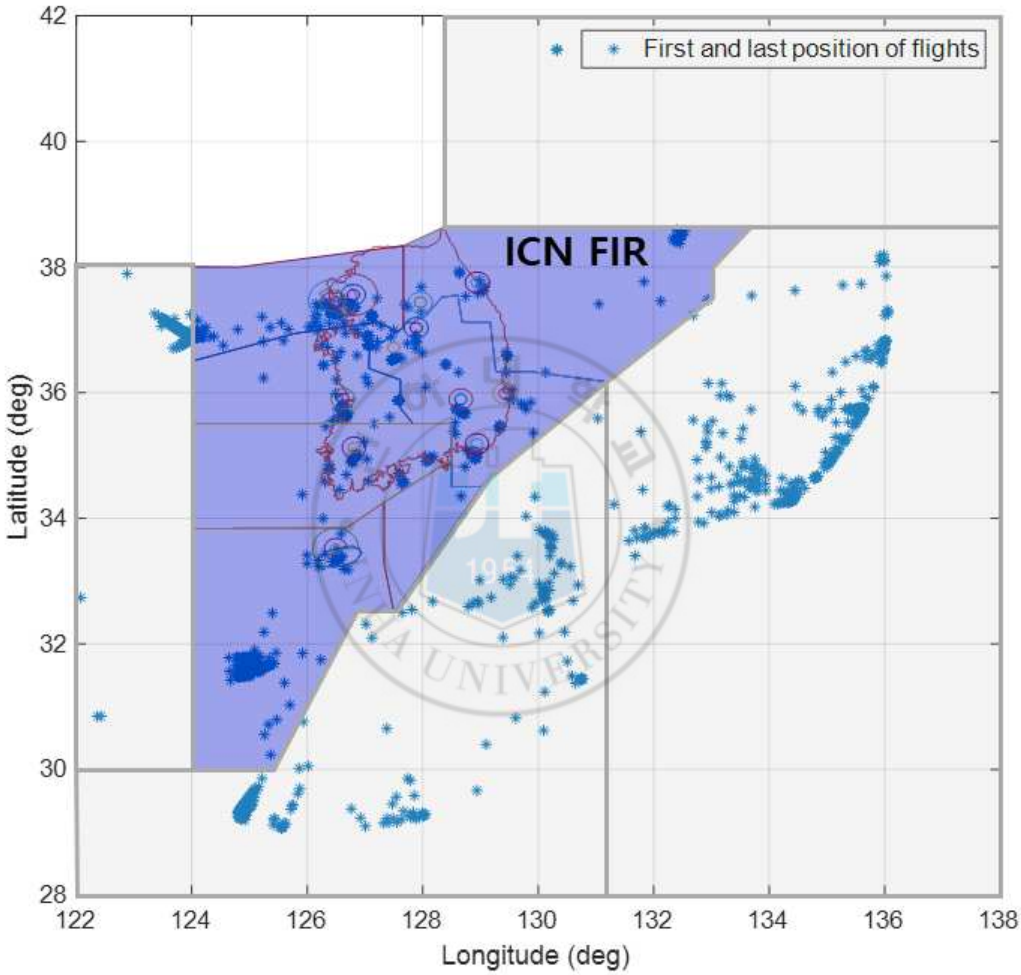
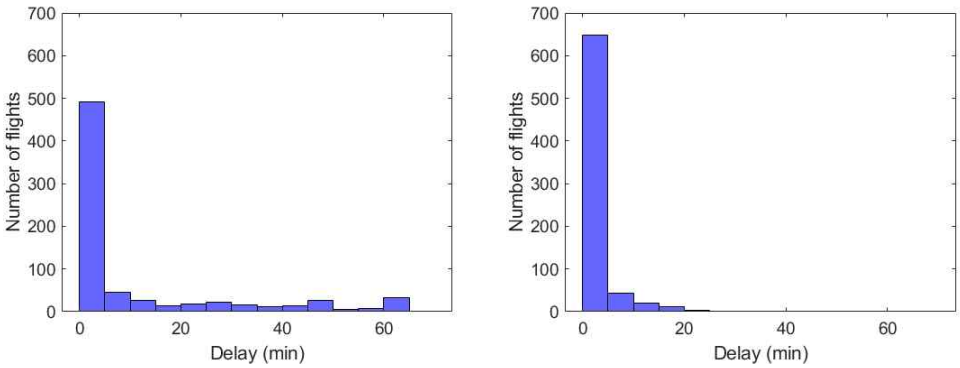


그림 3.4 대한민국 주변 공역 및 항적 데이터

위 가정을 통해, 국제선 항적의 전체적인 데이터가 없어도 인천 FIR 내부 공역과 공항의 제약 조건이 스케줄링에 끼치는 영향을 파악할 수 있다. 입력 스케줄은 약 730대의 항공기로 구성되었으며, 인천 FIR 내부에 존재하는 국내 공항의 ADR과 AAR은 모두 100으로 고정하였다. 그러나 인천 국제공항, 김포 국제공항, 제주 국제공항의 경우 ADR과 AAR을 50으로 줄여 스케줄링을 추가 수행하였다. 인천 FIR 내부 공역들은 최대 항공기 수용량을 10대로 고정하였다.

그림 3.5는 스케줄링을 통해 조정된 항공기들의 출도착 지연 시간 분포이다. 그림 3.5 (a)는 주요 3개 공항에 더 큰 제약을 주었을 때의 결과이며, 평균 지연 시간은 약 11분이다. 그림 3.5 (b)는 제약 조건에 여유를 주었을 때의 지연 시간 분포이며, 평균 지연 시간은 약 90초로 크게 감소하였다. 최대 지연 시간 역시 약 80분에서 20분으로 크게 감소하였으며, 이를 통해 교통량이 많은 주요 공항의 효율이 증가할수록 스케줄링 성능이 향상되는 것을 확인할 수 있다.

그림 3.6은 South West Sector의 항공기 수 변화이며, 그림 3.7은 인천 국제공항에 적용된 ADR, AAR 제약 조건과 스케줄링 결과의 ADR 및 AAR을 보여준다. 그림 3.7 (a), (b)는 최대 ADR, AAR이 50일 때의 결과이며, 그림 3.7 (c), (d)는 최대 ADR, AAR이 100일 때의 결과이다. 이로써 스케줄러가 주어진 공역 수용량과 출도착 노드의 제약 조건 모두 만족하는 결과를 제공하였음을 확인할 수 있다.



(a) Maximum ADR and AAR = 50 (b) Maximum ADR and AAR = 100

그림 3.5 항공기 출도착 지연 시간 분포

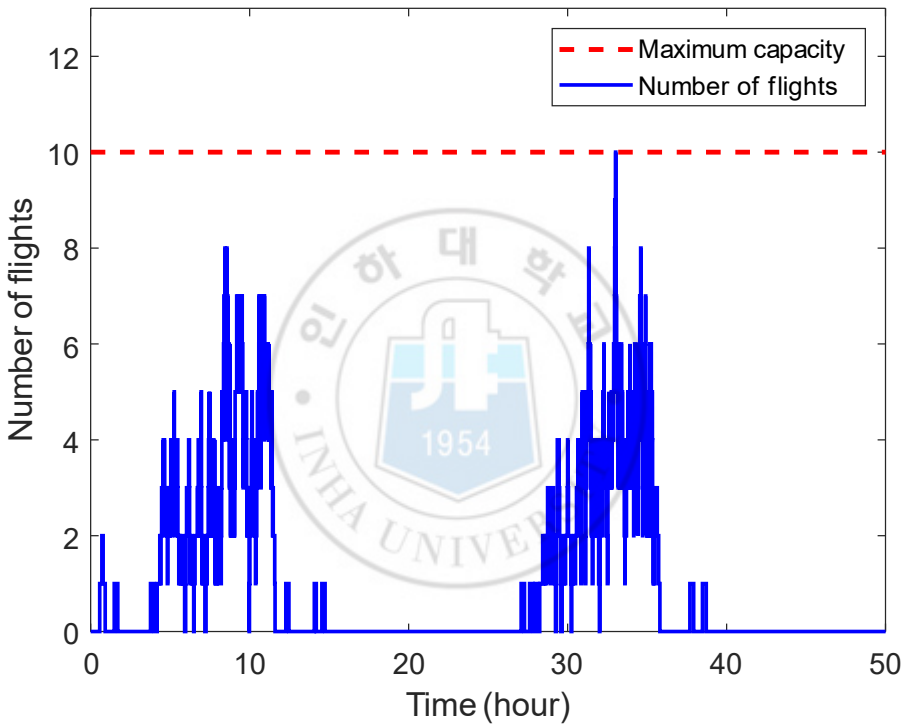
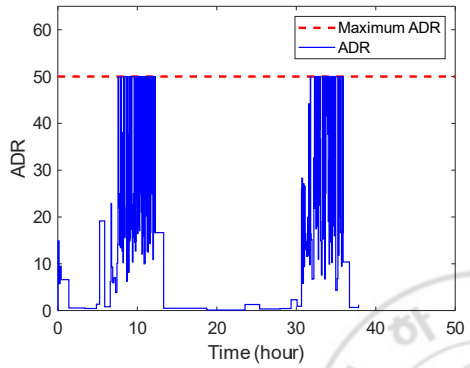
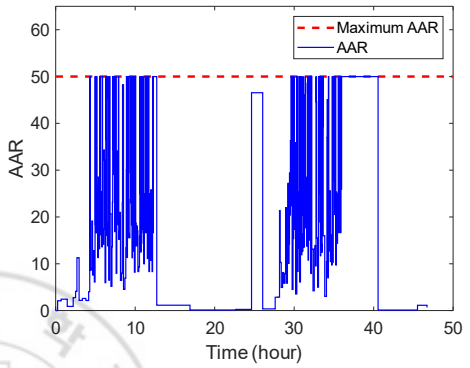


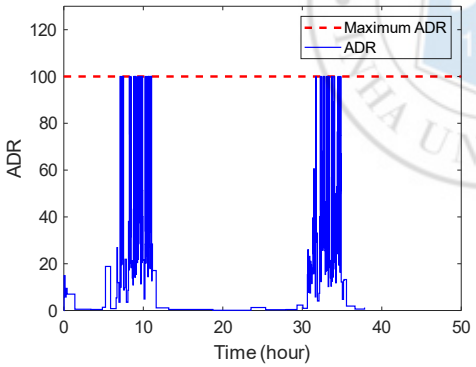
그림 3.6 South West Sector의 항공기 수 변화



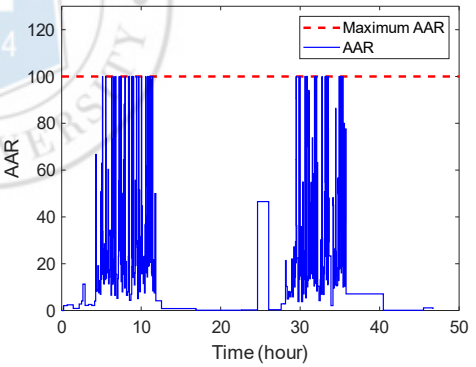
(a) Maximum ADR = 50



(b) Maximum AAR = 50



(c) Maximum ADR = 100



(d) Maximum AAR = 100

그림 3.7 인천 국제공항의 ADR 및 AAR 변화

4. 공항 스케줄링

본 장은 2장에서 설명한 FCFS 스케줄링 알고리즘을 공항 스케줄링에 적용한 '확장 FCFS(Extended First-Come First-Served, EFCFS) 스케줄링 알고리즘'에 대해 서술한다. 공항은 노드-링크의 제약 조건과 그 적용 방식이 공역과 다르다. 먼저 공항 노드-링크 모델의 특징을 설명하고, EFCFS 스케줄러는 공항 노드-링크의 제약 조건을 어떠한 형태로 적용하는지 상세히 설명한다. 또한, 다양한 시나리오를 기반으로 스케줄링을 수행하여 최적화 기반 알고리즘과 그 성능을 비교 분석한다.

4.1. 공항 노드-링크 아키텍처

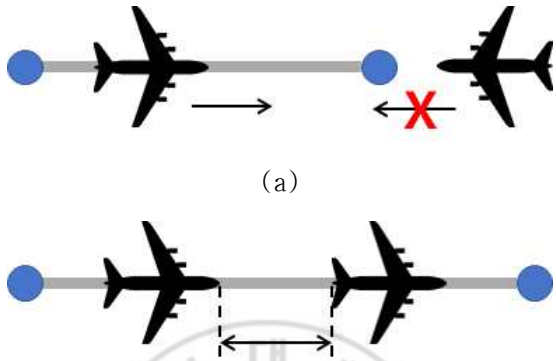
공항은 그 자체로 노드-링크 모델에 대응되며, 공항에서의 항공기 이동 경로는 공역과 달리 모든 노드와 링크가 위치, 길이 등의 물리적 제약이 존재한다. 따라서 공항 스케줄링의 경우 항공기가 지나는 모든 노드에서 제약 조건을 고려한다.

공항에서의 노드는 근본적으로 링크가 교차하는 지점이기 때문에, 한 항공기가 노드를 통과할 때 다른 항공기는 그 노드를 통과할 수 없다. 노드 제약 조건은 공역 스케줄링과 동일하게 노드 통과율을 적용한다. 항공기가 노드를 통과할 때, 항공기는 노드의 폭에 해당하는 거리를 이동하게 된다. 따라서 노드 통과율 r은 (4.1)과 같이 노드의 폭 w와 항공기 길이 l을 더한 값을 항공기 이동 속도 v로 나눈 값으로 정의되며, 노드 제약 조건 Δt는 r의 역수이다 [48]. 그러나 공항 활주로의 경우 '활주로 분리 기준(Runway separation criteria)'을 제약 조건으로 고려해야 한다.

$$\Delta t = \frac{1}{r},\ r = \frac{v}{w+l} \tag{4.1}$$

공항의 링크는 공역과 달리 그 길이가 정해져 있으며 항공기의 움직임이 제한적이다. 그림 4.1은 공항 링크 위에서의 항공기 움직임을 보여준다. 그림 4.1 (a)는 링크의 방

향성을 그림으로 나타낸 것이다. 항공기가 링크를 지나갈 때 다른 항공기가 맞은편에서 진입하는 경우, 링크 내 항공기가 링크를 완전히 통과할 때까지 해당 링크에 진입할 수 없다. 또한, 그림 4.1 (b)와 같이 항공기들은 서로 충돌하지 않도록 일정한 '분리 거리(Separation distance)'를 유지해야 한다.



(a)

(b)

그림 4.1 공항 링크에서의 항공기 움직임

4.1.1. 활주로 분리 기준 [49]

FCFS 스케줄링 알고리즘은 ADR, AAR과 같이 미리 정해진 값을 출도착 노드의 제약 조건으로 적용하였으나, 실제로 노드의 제약 조건은 선행(Leading) 및 후행(Trailing) 항공기의 중량 등급(Weight class) 혹은 '항적 난기류 범주(Wake turbulence category, WTC)'에 따라 결정되어야 한다. ICAO와 FAA는 항공기 이륙 중량을 기준으로 항적 난기류 범주와 이에 따른 '항공기 분리 기준(Aircraft separation criteria)'을 정의하였으며 [3, 50], 본 연구에서는 ICAO에서 정의한 시간 기반 분리(Time-based wake turbulence separation)를 적용하였다.

그림 4.2는 선행 항공기와 후행 항공기의 등급에 따라 노드 제약 조건이 결정되는 과정을 상세히 그린 것이다. 그림 4.2 (a)는 'Light(L)' 등급 항공기가 스케줄링 될 때, 이미 스케줄링이 완료된 항공기가 'Heavy(H)' 등급인 상황을 가정한 것이다. L 등급 항공기는 해당 노드에 H 등급 항공기보다 먼저 노드에 도착하거나, 이후에 도착하도록 스케줄링 될 수 있다. 이때 L 등급 항공기가 H 등급 항공기보다 먼저 도착할 경우 노

드 제약 조건은 Δt_{LH}이며, 나중에 도착할 경우 노드 제약 조건은 Δt_{HL}와 같다. 따라서 L 등급 항공기의 t_{EA}는 그림 4.2 (a)와 같이 회색으로 표시된 영역인 unavailable slot과 점선으로 표시된 영역인 entry slot의 비교를 통해 결정된다. 그림 4.2 (b)는 보다 복잡한 상황을 가정한 것으로, H, L 등급 항공기의 스케줄링이 완료된 상태에서 'Medium(M)' 등급 항공기가 스케줄링 되는 상황이다. 스케줄링 되는 항공기와 스케줄링이 완료된 항공기들의 위치 관계에 따라 노드 제약 조건이 결정되고, 이에 따라 해당 항공기의 t_{EA}가 결정되는 것을 확인할 수 있다.

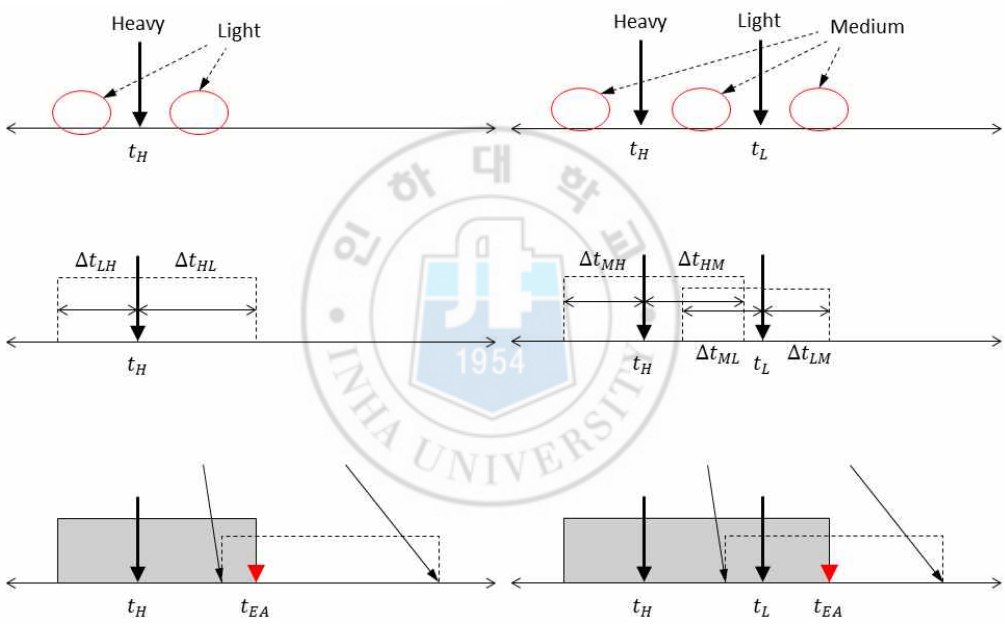


그림 4.2 활주로 분리 기준에 따른 노드 제약 조건

항공기들은 활주로의 '시단(Threshold)'과 '터치다운 구역(Touch down zone)'을 기점으로 이착륙을 수행한다. 따라서 시단과 터치다운 구역은 각각 출발 노드, 도착 노드와 동일하게 고려할 수 있다. 본 연구에서는 시단과 터치다운 구역을 따로 구분하지 않고 이착륙 모두 시단을 기점으로 이루어지도록 하였다. 그림 4.3은 활주로의 시단과 터치다운 구역을 보여준다.

항공기의 이착륙 과정에서 활주로 노드의 제약 조건은 그림 4.4와 같이 적용된다. 스케줄링을 통해 결정된 이륙 또는 착륙 시간이 t_0일 때, 실제 이착륙이 시작되는 시단과 활주로 맞은편의 시단은 그림 4.2와 같이 활주로 분리 기준을 제약 조건으로 적용한다. 양 시단을 제외한 나머지 활주로 노드의 경우, 이착륙 경로 상의 노드에만 항공기가 이착륙할 때 소요되는 '활주로 점유 시간(Runway occupancy time, ROT)'을 제약 조건으로 적용한다. 이는 항공기가 해당 노드를 통과하는 시간 이후 점유 시간이 끝날 때까지 다른 항공기들이 해당 활주로 노드에 진입할 수 없도록 한다.

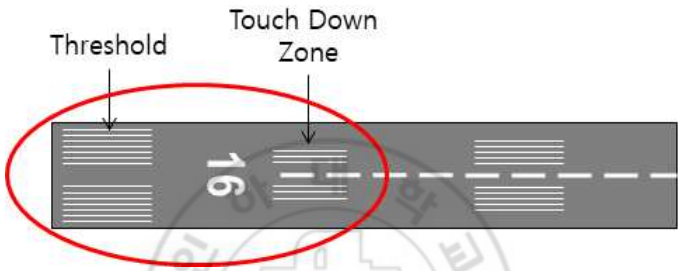


그림 4.3 활주로 시단과 터치다운 구역

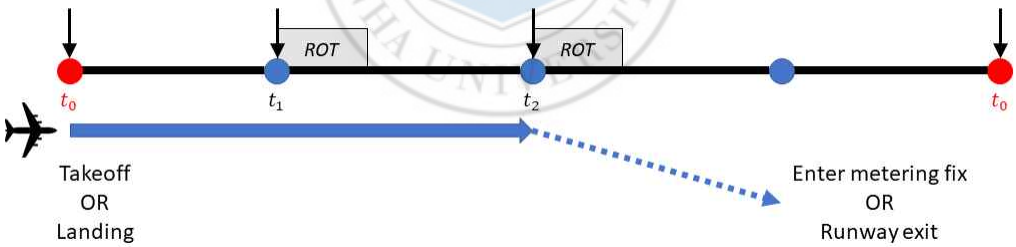


그림 4.4 활주로 이착륙 과정의 노드 업데이트

4.1.2. 링크 아키텍처

그림 4.1과 같이, 공항의 링크는 항공기의 이동 방향이 양방향으로 제한되어 있으며, 링크 위에서 항공기들이 서로 마주하거나 추월할 수 없다. 따라서 공역의 포화도를 제약 조건으로 적용하는 공역 링크 아키텍처와 달리, 공항 스케줄링의 링크 아키텍처는 항공기들의 충돌을 방지하기 위한 기준을 제약 조건으로 적용한다.

'Link blocking time'은 항공기 분리 거리를 분리 시간의 개념으로 정의한 것이다. Link blocking time은 항공기 1대가 링크를 진입하고 통과하는 시간 전후로 적용된다. 그림 4.5는 link blocking time 기반의 링크 제약 조건을 보여준다. 'Link transit block'은 항공기 1대가 링크에 진입한 뒤 완전히 통과할 때까지 다른 항공기들이 뒤에서 추월하거나 반대 방향에서 진입할 수 없도록 한다.

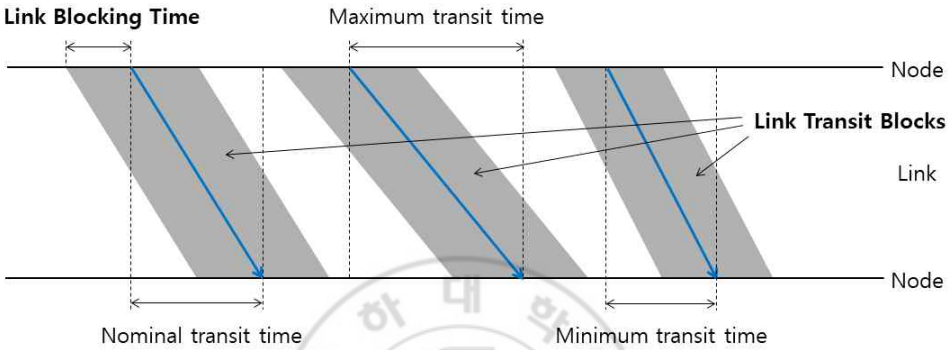
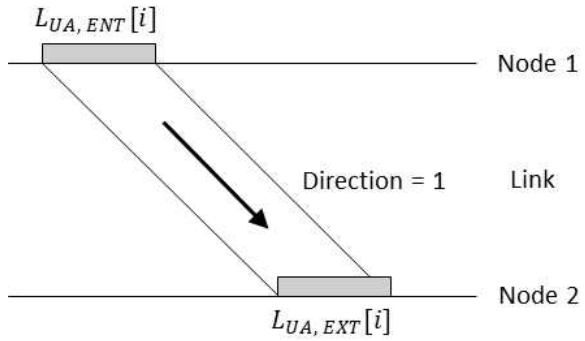


그림 4.5 Link blocking time 기반 링크 제약 조건

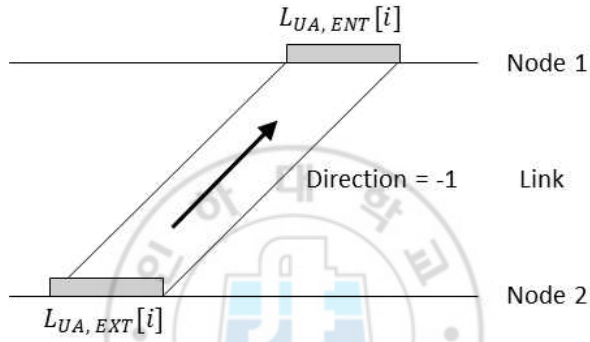
Link transit block은 그림 4.6과 같이 'Entry block', 'Exit block', 링크 이동 방향(Direction)으로 구성된다. 이동 방향은 링크의 주어진 기준 방향과 동일하면 1, 그 반대는 -1로 정의한다. Link entry block은 이동 방향과 관계없이 링크 기준 방향으로 진입하는 노드에 위치하며, link exit block은 기준 방향으로 통과하는 노드에 위치한다. (4.2), (4.3)은 각각 link entry block과 link exit block의 정의를 식으로 나타낸 것으로, t_{ENT}는 링크 진입 시간, t_{EXT}는 링크 통과 시간, t_{LB}는 link blocking time, t_L은 링크 이동 시간을 의미한다.

$$L_{UA,\,ENT}[i] = [t_{ENT} - t_{LB},\, t_{ENT} + t_{LB}] \tag{4.2}$$

$$L_{UA,\,EXT}[i] = [t_{EXT} - t_{LB},\, t_{EXT} + t_{LB}] = L_{UA,\,ENT}[i] \pm t_L \tag{4.3}$$



(a)



(b)

그림 4.6 항공기 이동 방향에 따른 링크 제약 조건

그림 4.7 (a), (b)는 각각 Forward 및 backward propagation 과정에서의 링크 제약 조건 처리 과정을 보여준다. (4.4), (4.5)는 이를 각각 식으로 나타낸 것이다. (4.4)에서 각 link entry block $L_{UA,\,ENT}[i]$가 Node 1의 available entry slot $N_{1_{AE\,f}}[j]$의 오른쪽에 위치할 경우 $k=0$, link entry block $L_{UA,\,ENT}[i]$가 $N_{1_{AE\,f}}[j]$의 왼쪽에 위치할 경우 $k=1$이다. (4.5)는 이와 반대로 각 link exit block $L_{UA,\,EXT}[i]$가 Node 2의 available entry slot $N_{2_{AE\,b}}[j]$의 오른쪽에 위치할 경우 $k=0$, link exit block $L_{UA,\,EXT}[i]$가 $N_{2_{AE\,b}}[j]$의 왼쪽에 위치할 경우 $k=1$이다.

$$N_{2_{AE,f}}[j] = \bigcap_{i=0}^{N} \left\{ \left(L_{UA,EXT}[i]^c \cap N_{2_{ENT,f}}[j] \right)[k] \right\},$$

$$where \;\; k = \begin{cases} 0, & \text{if } \; L_{UA,ENT}[i] \geq N_{1_{AE,f}}[j] \\ 1, & \text{if } \; L_{UA,ENT}[i] \leq N_{1_{AE,f}}[j] \end{cases} \tag{4.4}$$

$$N_{1_{AE,b}}[j] = \bigcap_{i=0}^{N} \left\{ \left(L_{UA,ENT}[i]^c \cap N_{1_{ENT,b}}[j] \right)[k] \right\},$$

$$where \;\; k = \begin{cases} 0, & \text{if } \; L_{UA,EXT}[i] \geq N_{2_{AE,b}}[j] \\ 1, & \text{if } \; L_{UA,EXT}[i] \leq N_{2_{AE,b}}[j] \end{cases} \tag{4.5}$$

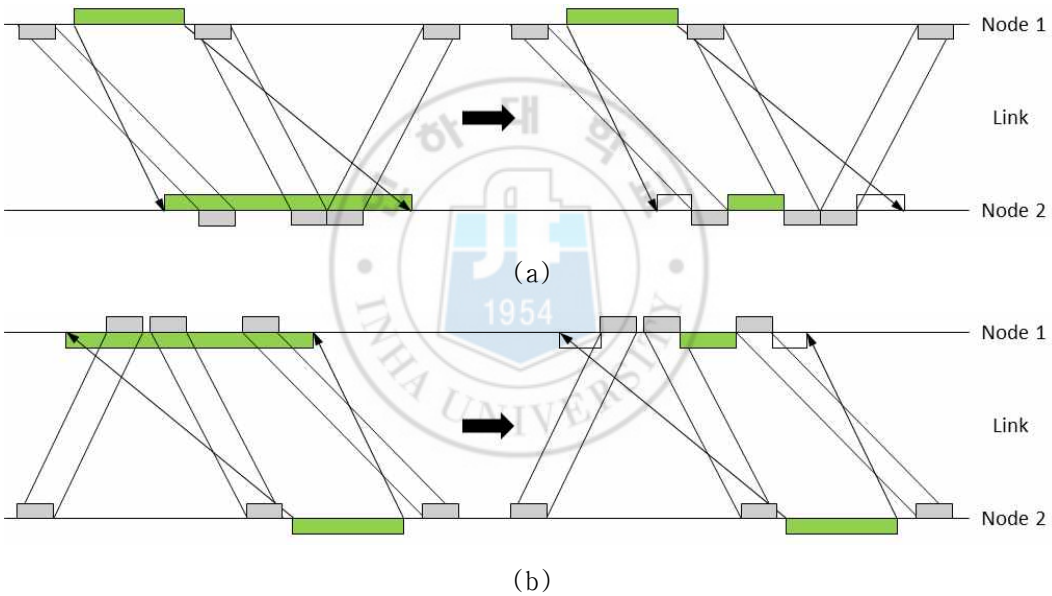


그림 4.7 Forward 및 Backward propagation 과정에서의 링크 제약 조건 처리

그림 4.8은 공항 링크 아키텍처와 링크를 지나는 항공기 대수 변화를 보여준다. 앞서 정의한 link blocking time과 link transit block으로 인해 항공기 간의 충돌이나 추월이 발생하지 않는 것을 확인할 수 있다. 그림 4.9, 4.10은 각각 공항 링크 아키텍처가 적용된 EFCFS 스케줄링 알고리즘의 propagation 결과 및 스케줄링 결과 예시이다. 링크 제약 조건이 공역 스케줄링과 달리 평행사변형의 형태임을 확인할 수 있다.

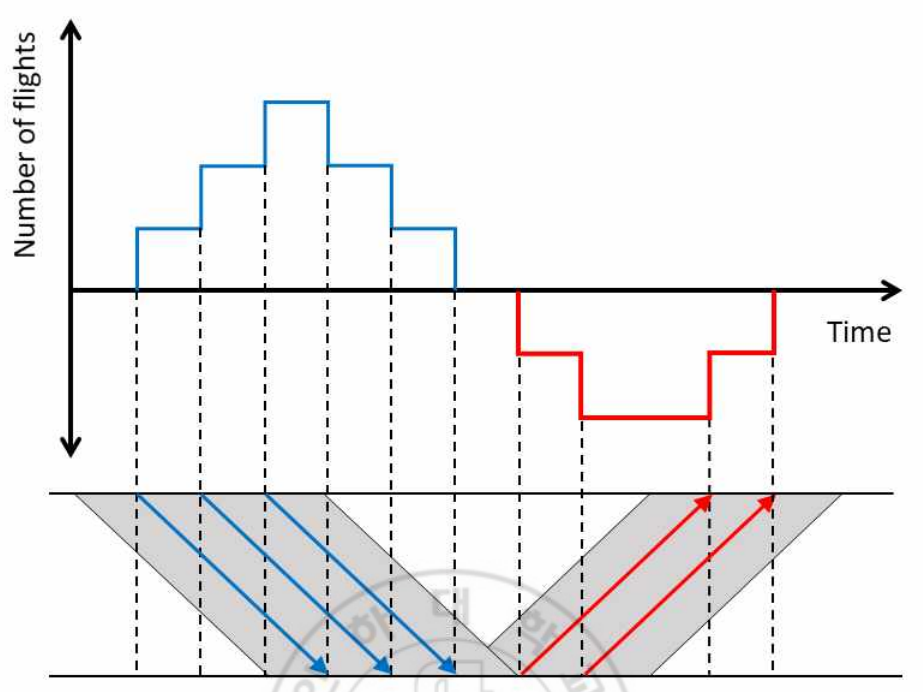


그림 4.8 공항 링크 아키텍처

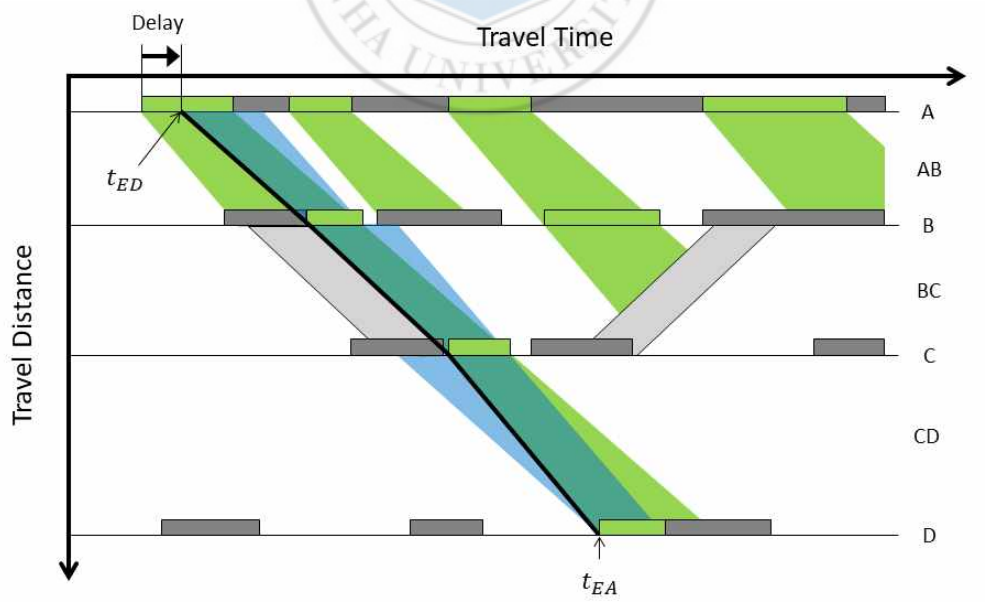


그림 4.9 EFCFS 스케줄러의 propagation 결과 예시

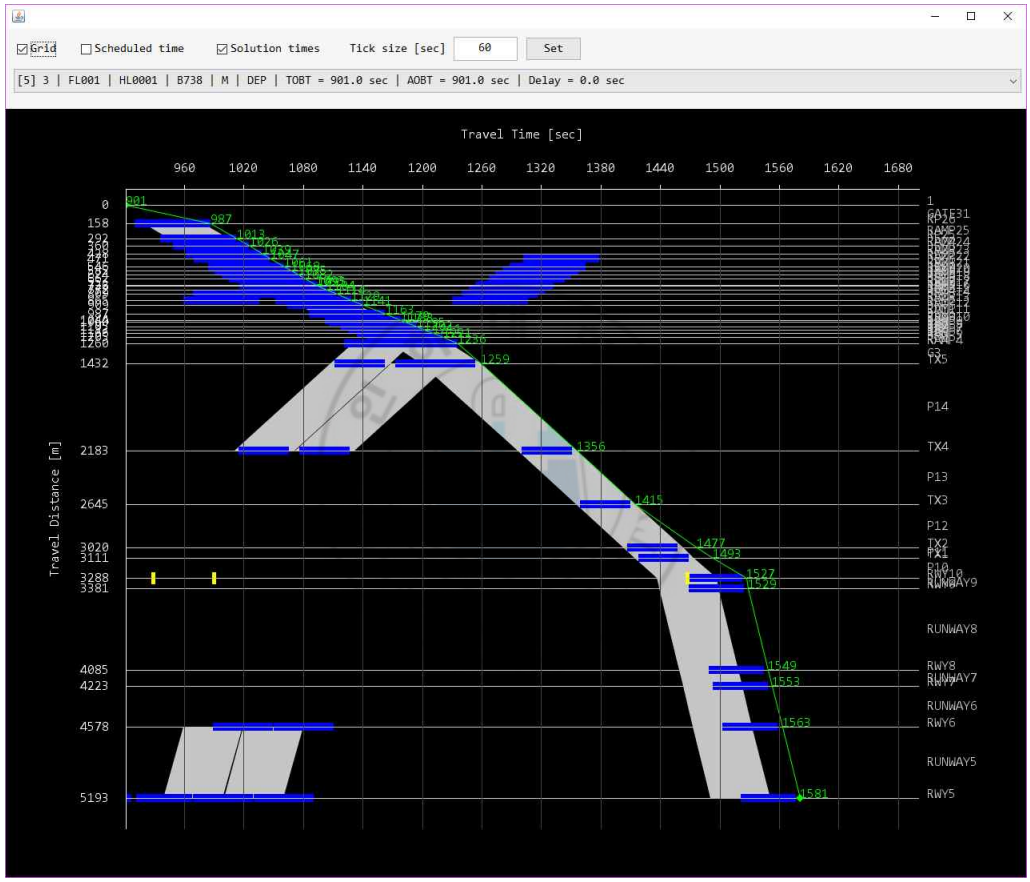


그림 4.10 EFCFS 스케줄러의 스케줄링 결과

4.2. 경로 할당 (Route assignment) [48, 49]

항공기 출도착 관리의 목표 중 하나는 항공기가 효율적인 경로로 이동하도록 하는 것이다. 따라서 본 연구에서는 EFCFS 스케줄러의 장점 중 하나인 빠른 계산 속도를 활용하여 다양한 항공기 이동 경로를 탐색하였다.

일반적으로 이동 경로는 목적에 따라 다양하게 탐색할 수 있으나, 가장 기본적인 목적은 '이동 거리' 또는 '이동 시간'의 최소화이다. 'Dijkstra 알고리즘 [51]'은 주어진 노드-링크 모델에서 최소 비용 경로를 결정하는 대표적인 기법이다. 'K-Shortest-Paths 알고리즘 [52]'은 Dijkstra 알고리즘을 기반으로 여러 개의 최단 경로를 탐색하는 기법이다. 먼저 Dijkstra 알고리즘으로 최단 경로를 결정한 뒤, 최단 경로를 기준으로 두 번째 최단 경로를 탐색한다. 이후 K번째 최단 경로를 탐색할 때까지 이를 반복한다.

본 연구에서는 이 기법을 활용해 항공기당 최대 k개의 이동 경로들을 탐색하였다. 활주로 이착륙 경로는 단일 경로이므로, 항공기의 이동 경로 탐색은 그림 4.11과 같이 게이트와 활주로 시단 사이의 범위 내에서 수행된다. EFCFS 스케줄러는 각 항공기마다 모든 이동 경로에 대해 스케줄링을 수행한 뒤 출발 노드에서 최소 지연 시간(Minimum delay)을 가지는 경로를 선택한다. 최소 지연 경로는 이동 거리가 아닌 스케줄링을 수행할 때 대상 항공기의 상황에 따라 결정된다.

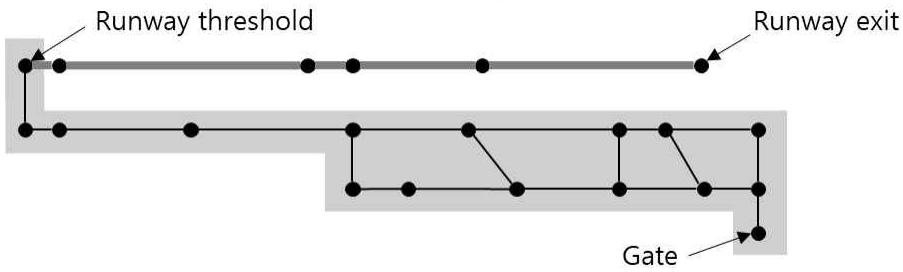


그림 4.11 공항 노드-링크 경로 탐색 범위

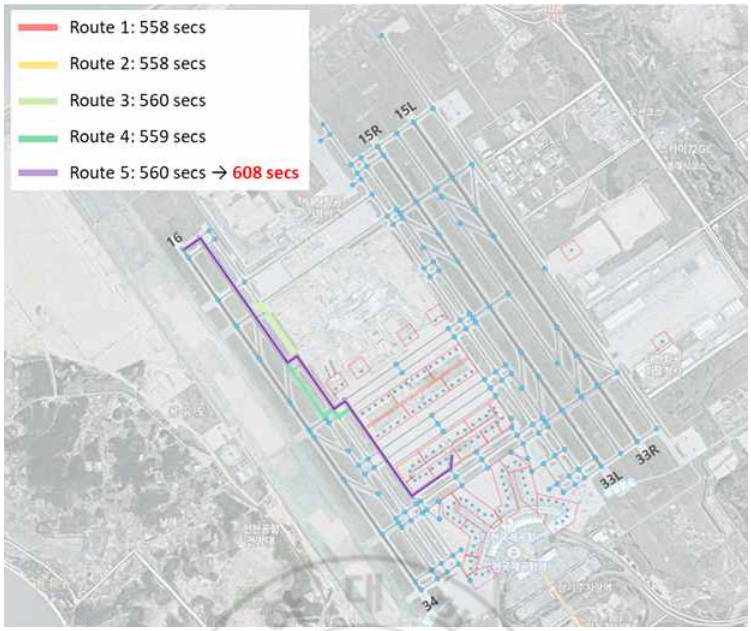
그림 4.12는 인천 국제공항을 대상으로 경로 할당을 수행한 결과이다. 입력 스케줄은 2015년 4월 1일 FOIS 데이터를 기반으로 하며, 399대의 출발 항공기와 385대의 도착 항공기로 구성되었다. 계류장(Ramp, Apron) 구역은 여러 개의 게이트를 한 블록에 묶

어 단순화하였으며, 화물 터미널은 2개의 노드로 집약하였다. 경로 할당의 후보 경로의 수는 최대 5개로 설정하였으며, 항공기의 이동 속도 변화율은 최대 5% 가속, 10% 감속으로 제한하였다.

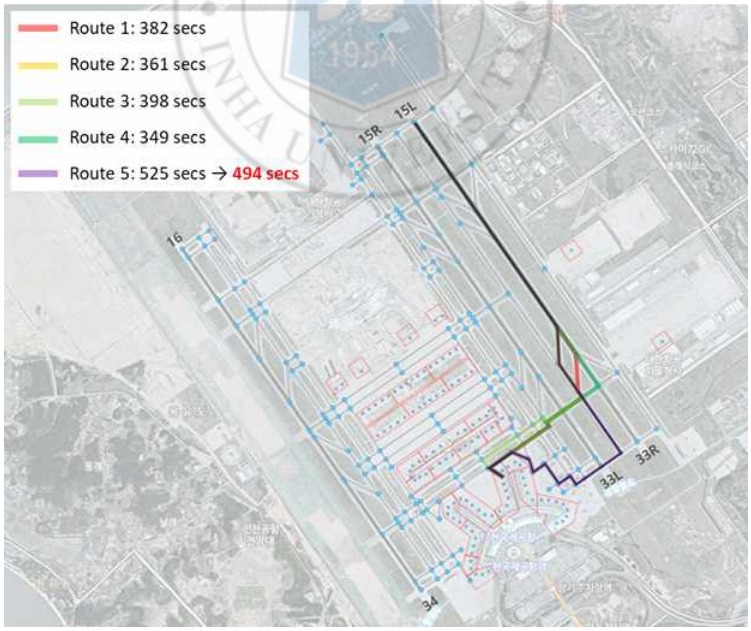
공항에서의 항공기 지연 시간은 총 4가지로 구분하여 정의된다. 출발 항공기의 지연 시간은 '게이트 출발 지연 시간(Delay of Off-Block Time, DOBT)' 및 '활주로 이륙 지연 시간(Delay of Take-Off Time, DTOT)'으로 구분되며, 도착 항공기의 지연 시간은 '활주로 착륙 지연 시간(Delay of Landing Time, DLDT)' 및 '게이트 도착 지연 시간(Delay of In-Block Time, DIBT)'으로 구분된다. 각 지연 시간은 (4.6)과 같이 '원래 예정된(Scheduled)' 시각과 스케줄러에서 '계산된(Target)' 시각의 차이로 정의된다. 여기서 STOT는 SOBT와 XOT의 합, SIBT는 SLDT와 XIT의 합으로 정의된다. 이때 XOT는 게이트를 출발한 뒤 아무 제약 없이 유도로를 이동해 활주로에서 이륙하기까지의 시간(Unimpeded taxi-out time)을 의미하며, XIT는 활주로에 착륙하여 아무 제약 없이 유도로를 이동하여 게이트에 도착할 때까지의 시간(Unimpeded taxi-in time)을 의미한다.

$$
\begin{aligned}
DOBT &= TOBT - SOBT \\
DTOT &= TTOT - STOT = TTOT - (SOBT + XOT) \\
DLDT &= TLDT - SLDT \\
DIBT &= TIBT - SIBT = TLDT - (SLDT + XIT)
\end{aligned}
\tag{4.6}
$$

그림 4.12는 경로 할당 결과로, 스케줄러가 최단 거리 경로가 아닌 다른 이동 경로를 최소 지연 경로로 선택하였음을 확인할 수 있다. 그림 4.12 (a)는 활주로 16에서 출발하는 항공기의 이동 경로이며, 총 5개의 후보 경로 중 일반적인 이동 시간이 560초로 가장 긴 5번 경로가 선택되었다. 5번 경로의 이동 시간이 608초로 증가했음에도 불구하고 이러한 결과가 나온 이유는 스케줄러가 경로 할당을 통해 DOBT가 가장 작은 경로를 선택하였기 때문이다. 그림 4.12 (b)는 활주로 15L에 도착하는 항공기의 이동 경로를 보여준다. 이 역시 이동 시간이 525초로 가장 긴 5번 경로가 선택되었으며, 이 경우 최대한 빠른 속도로 이동하여 이동 시간이 494초로 감소하였다.



(a) 활주로 16 출발 항공기 경로



(b) 활주로 15L 도착 항공기 경로

그림 4.12 인천 국제공항에서의 항공기 경로 할당 결과 예시

4.3. 스케줄링 결과

4.3.1. 스케줄링 우선순위 분석 [49]

스케줄링 우선순위에 따른 결과를 비교하기 위해, 인천 국제공항을 대상으로 스케줄링을 수행하였다. 입력 스케줄은 총 3개로, 각각 2015년 4월 1일, 3일, 10일에 기록된 FOIS(Flight Operation Information System) 데이터를 기반으로 구성하였다. 제 2 터미널은 고려하지 않았으며, 활주로 분리 기준은 부록 A의 표 A.1~A.5를 적용하였다.

스케줄링 우선순위는 3가지로 고려되었다. 'Nominal priority'는 항공기마다 원래 예정된 출도착 시간을 기준으로 한다. 즉, 예정된 출도착 시간이 빠른 항공기를 우선하여 스케줄링하는 방식이다. 'Arrival priority'는 모든 도착 항공편을 먼저 스케줄링한다. 이 때 도착 항공편 간의 우선순위와 출발 항공편 간의 우선순위는 nominal priority로 적용된다. 마지막으로, 'Partial arrival priority'는 일정 시간 구간마다 arrival priority를 적용하는 방식으로, 일정 시간 내에 도착 항공편을 먼저 계산한 뒤 출발 항공편을 스케줄링하며, 이를 구간마다 반복한다.

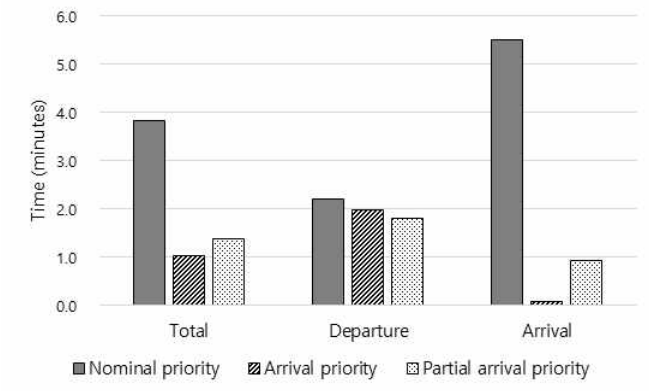
표 4.1, 4.2는 각각 우선순위에 따른 스케줄링 결과의 평균 지연 시간 및 최대 지연 시간을 정리한 것이며, 그림 4.13은 표 4.1을 그래프로 나타낸 것이다. 출발 항공기는 DOBT, 도착 항공기의 경우 DLDT를 지연 시간으로 고려하였다. Arrival priority 및 partial arrival priority를 적용한 경우, 도착 항공편의 DLDT가 크게 감소하여 전체적인 평균 지연 시간이 감소한 것을 확인할 수 있다. 그러나 arrival priority는 모든 도착 항공편을 먼저 스케줄링하는 특성으로 인해 DOBT가 오히려 증가하는 역효과가 발생하며, 이를 구간별 반복 수행하는 partial arrival priority는 이러한 역효과가 억제된 결과를 보여준다.

표 4.1 스케줄링 우선순위에 따른 평균 지연 시간 (단위: 분)

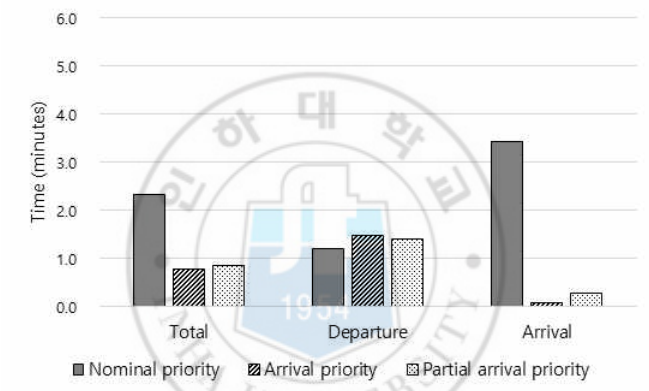
Date	Flights		Nominal priority	Arrival priority	Partial arrival priority
2015.04.01	Total	782	3.8	1.0	1.4
	Departure	397	2.2	2.0	1.8
	Arrival	385	5.5	0.1	0.9
2015.04.03	Total	717	2.3	0.8	0.8
	Departure	361	1.2	1.5	1.4
	Arrival	356	3.4	0.1	0.3
2015.04.10	Total	821	2.9	1.4	1.3
	Departure	405	2.0	2.8	2.0
	Arrival	416	3.7	0.1	0.4

표 4.2 스케줄링 우선순위에 따른 최대 지연 시간 (단위: 분)

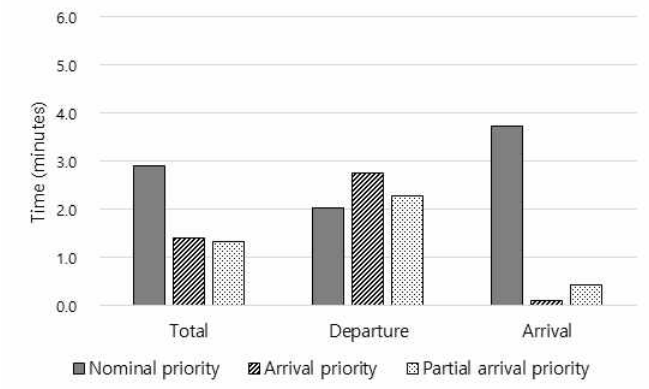
Date	Flights		Nominal priority	Arrival priority	Partial arrival priority
2015.04.01	Departure	397	21.6	26.0	26.0
	Arrival	385	23.8	2.0	11.0
2015.04.03	Departure	361	13.8	17.4	13.6
	Arrival	356	13.1	3.0	9.9
2015.04.10	Departure	405	18.9	43.9	19.6
	Arrival	416	16.3	2.0	13.3



(a) 2015.04.01



(b) 2015.04.03



(c) 2015.04.10

그림 4.13 스케줄링 우선순위에 따른 평균 지연 시간

4.3.2. 최적화 기반 알고리즘과의 비교 [53]

개선된 EFCFS 스케줄러의 성능을 검증하기 위해, MILP 기반 스케줄러 [23]와 스케줄링 결과를 비교 분석하였다. 타당한 비교를 위해, 다음과 같이 두 스케줄링 기법의 제약 조건들과 스케줄링 방식을 조정하였다.

EFCFS 스케줄러는 게이트부터 활주로까지 모든 과정을 한 번에 스케줄링한다. 반면, MILP 스케줄러는 활주로 스케줄링을 통해 활주로 분리 기준을 만족하도록 항공기 이착륙 시간이 먼저 조정된다. 즉, 착륙 시간이 미리 계산되므로, MILP 모델은 도착 항공기에 대해 택시 스케줄링만을 수행한다. EFCFS 스케줄러는 arrival priority를 통해 착륙 시간이 미리 정해지도록 하였다.

두 스케줄링 기법이 공통적으로 고려한 제약 조건은 표 4.3과 같다. 노드 제약 조건은 출발 항공기가 가능한 빨리 게이트에서 출발할 수 있도록 하며, WTC에 따른 활주로 분리 기준과 Miles-In-Trails(MIT)를 고려하여 활주로와 'Metering fix'에서의 최소 분리 시간을 적용한다. 링크 제약 조건은 동일한 링크에서 항공기들이 서로 마주하거나 추월하는 일이 없도록 한다.

표 4.3 EFCFS와 MILP 스케줄러의 공통 제약 조건

Type	Constraints
Node constraints	Earliest possible off-block times
	Runway separation criteria
	Miles-In-Trails(MIT)
Link constraints	No deadlock in bi-directional taxiway links
	Aircraft separation along the taxiways

이때, MIT 제약 조건은 EFCFS 스케줄러에 입력되는 노드-링크 모델을 공항 활주로에서 metering fix까지 확장하고, 활주로 노드에서 metering fix까지 propagation 과정을 추가로 수행하면서 자연스럽게 적용할 수 있다. 그림 4.14는 departure fix 노드가 추가된 노드-링크 모델의 예시이다. MIT 제약은 그림 4.15와 같이 시간 블록으로 적용된다. 블록의 크기는 (4.7)과 같이 MIT 거리 D_{MIT}를 metering fix에서의 항공기 속

도 V_{Fix}로 나눈 값으로 결정된다.

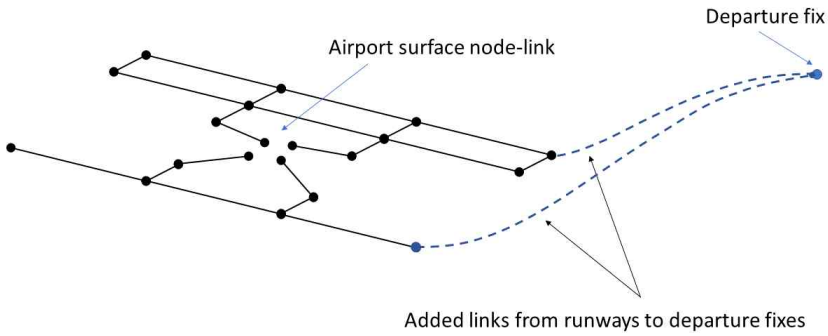


그림 4.14 Departure fix까지 확장된 노드-링크 모델 예시

$$\Delta t = \frac{D_{MIT}}{V_{Fix}} \tag{4.7}$$

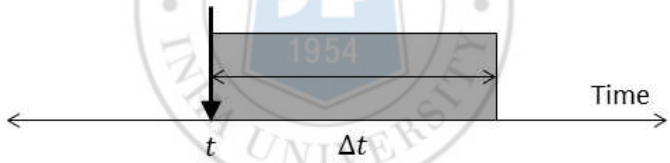


그림 4.15 Metering fix에서의 MIT 제약 조건

입력 스케줄은 인천 국제공항에서 1시간 동안 출도착하는 60대의 항공기들로 구성되었으며, 전후 교통 상황 및 제 2 터미널은 고려되지 않았다. 항공기들의 WTC 구성비는 표 4.4와 같다. 그림 4.16은 인천 국제공항의 노드-링크 모델과 출발 Fix를 보여준다. 도착 항공기들은 모두 활주로 33R에 착륙하며, 출발 항공기들은 활주로 33L 및 34를 모두 사용한다. 표 4.5는 출발 항공기들이 할당된 활주로와 출발 Fix를 정리한 것이다. 활주로 분리 기준은 부록 A의 표 A.1~A.5를 적용하였다. 출발 Fix 중 South 및 West Fix만 MIT 제약 조건을 고려하였으며, MIT는 15 nmi(해리, nautical miles)로 설정하였다.

표 4.4 항공기 WTC 구성

	Departure	Arrival
Medium	14	7
Heavy	26	13

표 4.5 활주로 및 출발 Fix에 할당된 출발 항공기

Departure fixes	Runways	Flights	MIT
East	15R/33L	5	–
South East		5	–
South	15R/33L	6	O
	16/34	9	
West	16/34	15	O

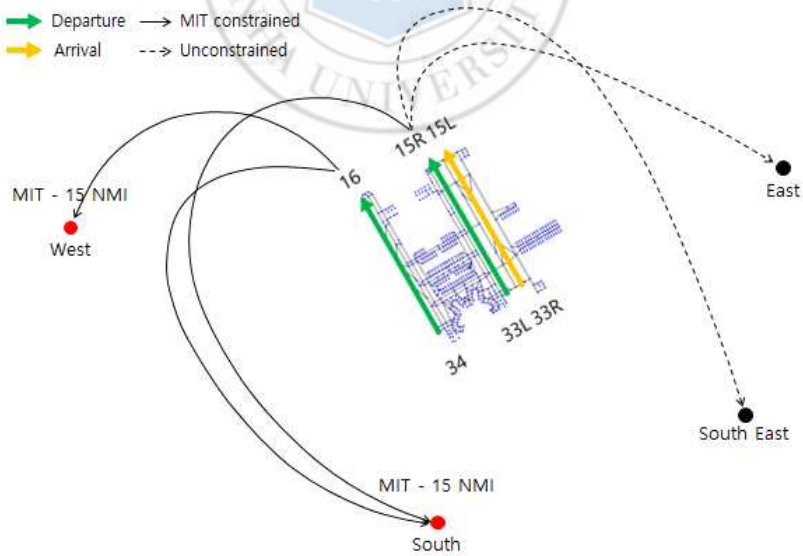


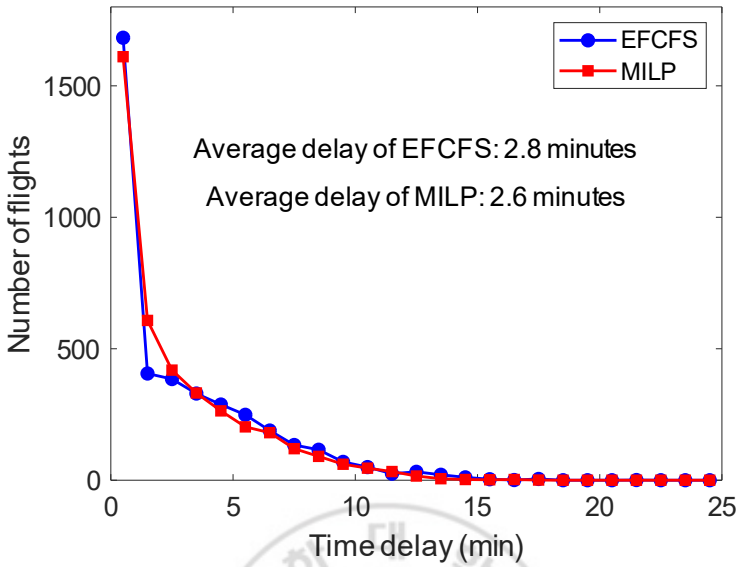
그림 4.16 인천국제공항 노드-링크 및 출발 Fix

총 100개의 입력 시나리오를 생성해 스케줄링을 수행하였으며, 각 시나리오의 출발 항공편의 게이트 및 SOBT, 도착 항공편의 SLDT는 모두 임의로 지정되었다. SLDT의 경우 활주로 분리 기준에 만족하도록 미리 계산되었기 때문에 스케줄러가 따로 조정하지 않는다.

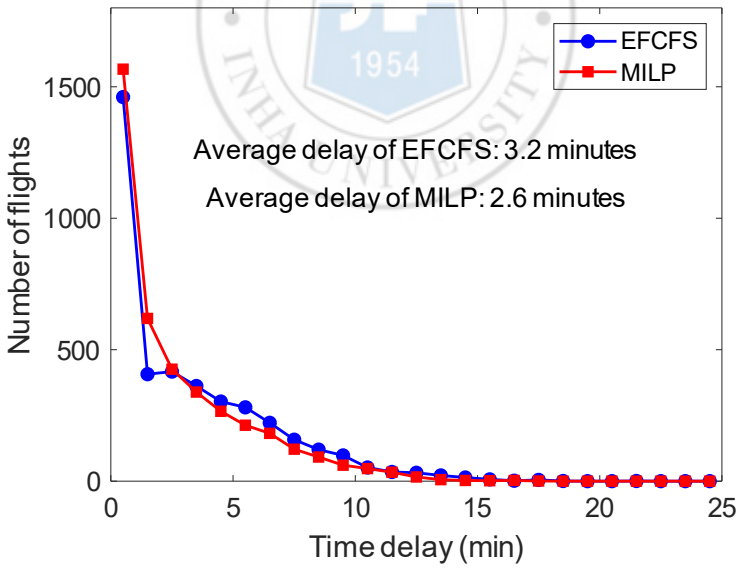
그림 4.17 ~ 4.19는 MIT 제약 조건을 고려하지 않고 스케줄링을 수행하였을 때의 결과이다. 그림 4.17은 100개 시나리오를 모두 고려한 출발 항공기 4000대의 지연 시간 분포를 보여준다. EFCFS와 MILP의 평균 DOBT는 각각 2.8분, 2.6분이며, 평균 DTOT는 각각 3.2분, 2.6분이다. MILP는 평균 DOBT와 DTOT의 차이가 없으며, EFCFS는 택시 지연이 발생하여 DTOT가 증가한 것을 확인할 수 있다.

그림 4.18은 모든 시나리오에 대해, 각 시나리오에서 출발 항공기의 최대 지연 시간을 추출하여 그 분포를 그린 것이다. EFCFS와 MILP의 평균 최대 DOBT는 각각 10.9분, 9.8분이며, 평균 최대 DTOT는 각각 11.2분, 9.8분이다. 최대 지연 시간 역시 MILP가 EFCFS보다 약간 더 좋은 결과를 보여준다.

그림 4.19는 100개 시나리오에 대한 EFCFS와 MILP 스케줄러의 makespan 차이의 분포이다. 양의 방향은 EFCFS의 makespan이 MILP보다 크다는 것을 의미한다. EFCFS와 MILP의 평균 makespan은 각각 89.5분, 89분으로 MILP가 EFCFS에 비해 약간 좋은 성능을 보인다.

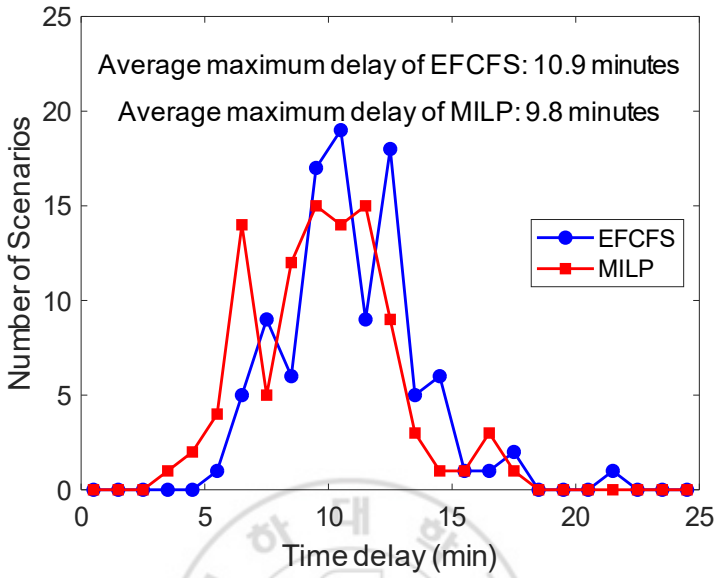


(a) DOBT

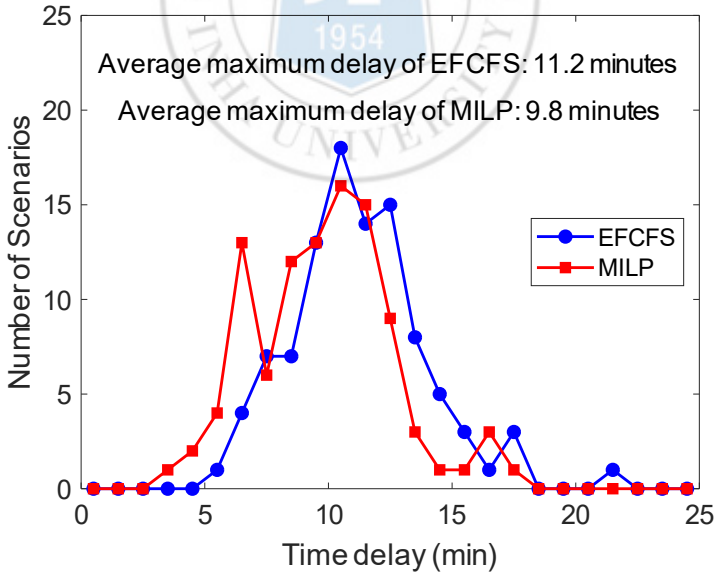


(b) DTOT

그림 4.17 100개 시나리오의 출발 항공기 지연 시간 분포



(a) 최대 DOBT



(b) 최대 DTOT

그림 4.18 100개 시나리오의 출발 항공기 최대 지연 시간 분포

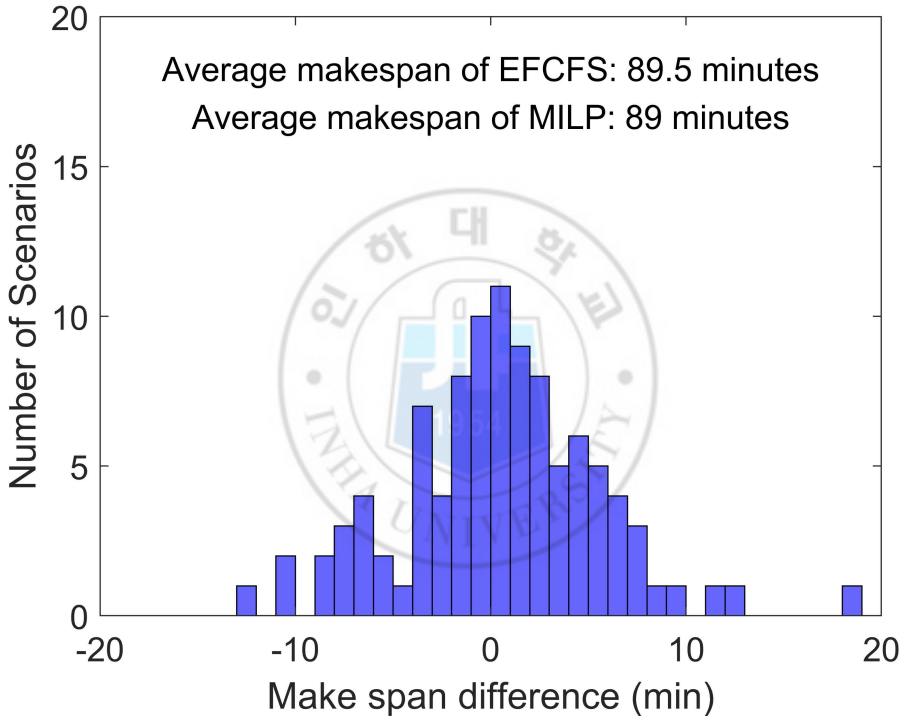


그림 4.19 100개 시나리오에 대한 EFCFS와 MILP 스케줄러의 makespan 차이

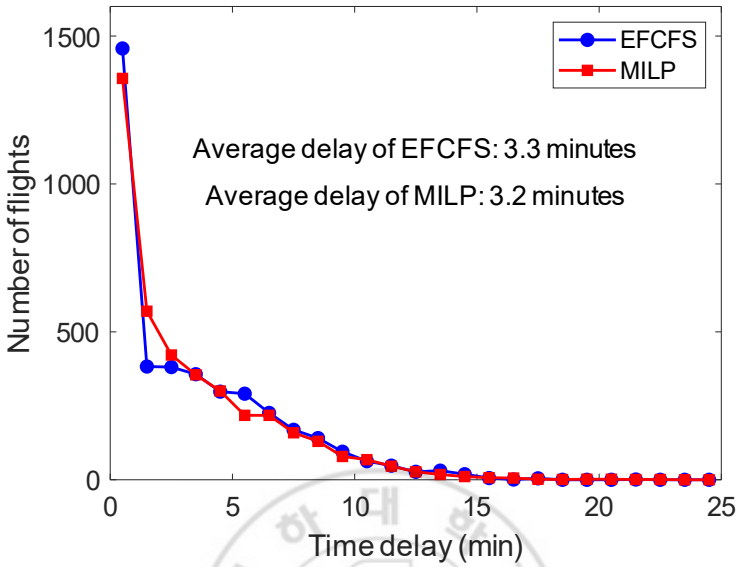
그림 4.20 ~ 4.22는 MIT 제약 조건을 고려하여 스케줄링을 수행하였을 때의 결과로, MIT는 출발 Fix 중 South 및 West Fix에 적용되었다. 그림 4.20은 출발 항공기들의 지연 시간 분포를 보여준다. EFCFS는 3.3분, MILP는 3.2분의 평균 DOBT를 가지며, 평균 DTOT의 경우 EFCFS는 3.7분, MILP는 3.3분이다. MIT가 적용되지 않았을 때와 비교하여, MILP의 지연 시간이 EFCFS에 비해 더 크게 증가하였다.

그림 4.21은 모든 시나리오의 최대 출발 지연 시간 분포이다. EFCFS와 MILP의 평균 최대 DOBT는 각각 11.6분, 11.7분이며, 평균 최대 DTOT는 각각 11.9분, 11.8분이다. 최대 지연 시간의 경우, 두 기법의 차이는 크지 않다.

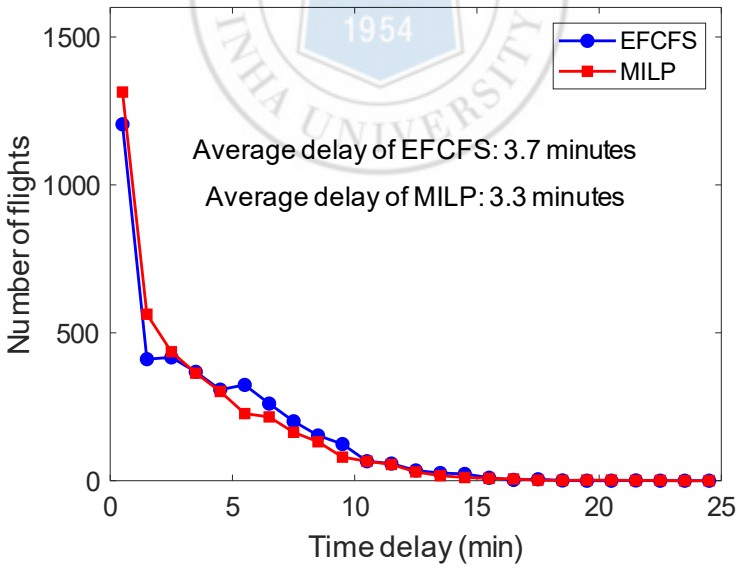
그림 4.22는 두 기법의 makespan 차이의 분포를 보여주며, EFCFS와 MILP의 평균 makespan은 각각 90.2분, 89.7분으로 이 역시 큰 차이가 없다.

그림 4.23는 두 스케줄링 기법의 각 시나리오별 계산 시간을 비교한 것이다. 그림 4.23 (a)는 MIT를 고려하지 않고 스케줄링을 수행하였을 때의 계산 시간을 보여주며, EFCFS와 MILP는 각각 평균 0.82초, 6.39초의 계산 시간이 소요되었다. 그림 4.23 (b)는 MIT 제약 조건을 고려했을 때의 스케줄링 계산 시간이다. EFCFS와 MILP의 평균 계산 시간은 각각 0.99초, 9.22초이며, EFCFS는 계산 시간이 거의 증가하지 않은 반면 MILP는 계산 시간이 크게 증가하였다.

전체적으로 MILP 기법이 약간 좋은 성능을 보이지만 두 기법의 차이가 크지 않으며, MILP의 경우 오히려 MIT 제약 조건이 추가되면서 효율이 감소되었음을 알 수 있다. 또한, 동일한 문제에 대해 EFCFS 기법이 MILP보다 약 10배 빠른 계산 속도를 보이며, 이 차이는 문제 사이즈 및 제약 조건이 증가할수록 더욱 커질 수 있다. 따라서 EFCFS 스케줄러가 문제의 사이즈 및 제약 조건에 관계없이 빠른 계산 속도를 유지하면서도 MILP와 거의 유사한 성능을 가지고 있음을 확인하였다.

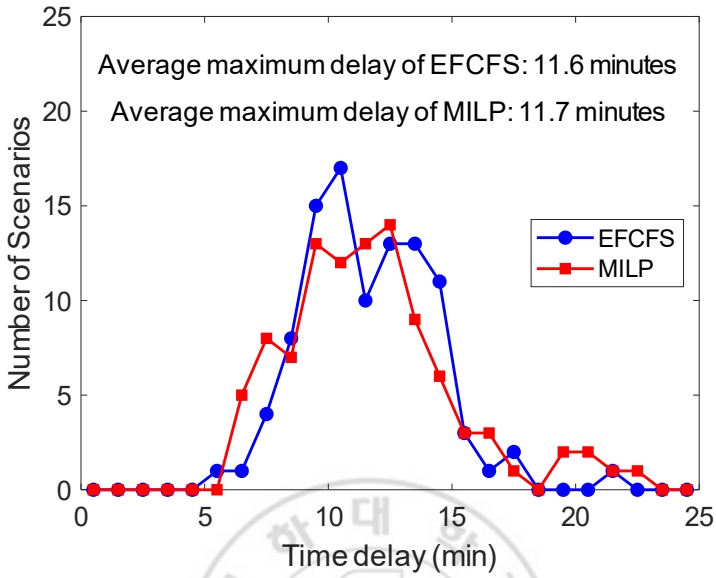


(a) DOBT

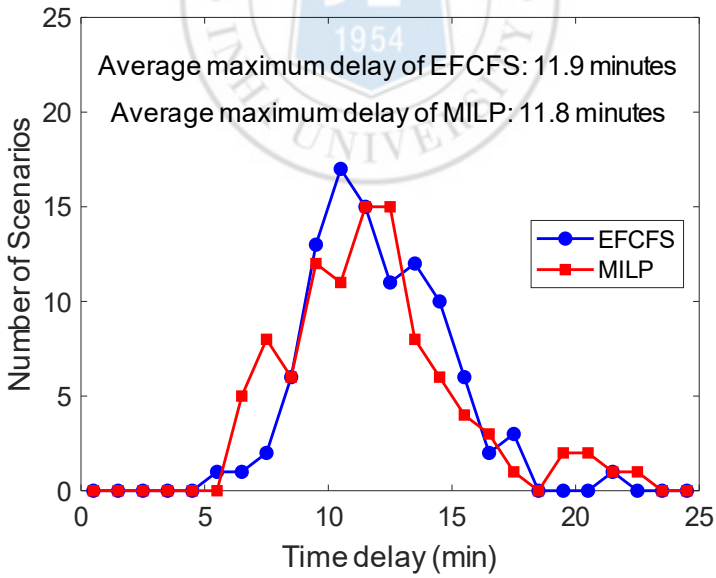


(b) DTOT

그림 4.20 100개 시나리오의 출발 항공기 지연 시간 분포



(a) 최대 DOBT



(b) 최대 DTOT

그림 4.21 100개 시나리오의 출발 항공기 최대 지연 시간 분포

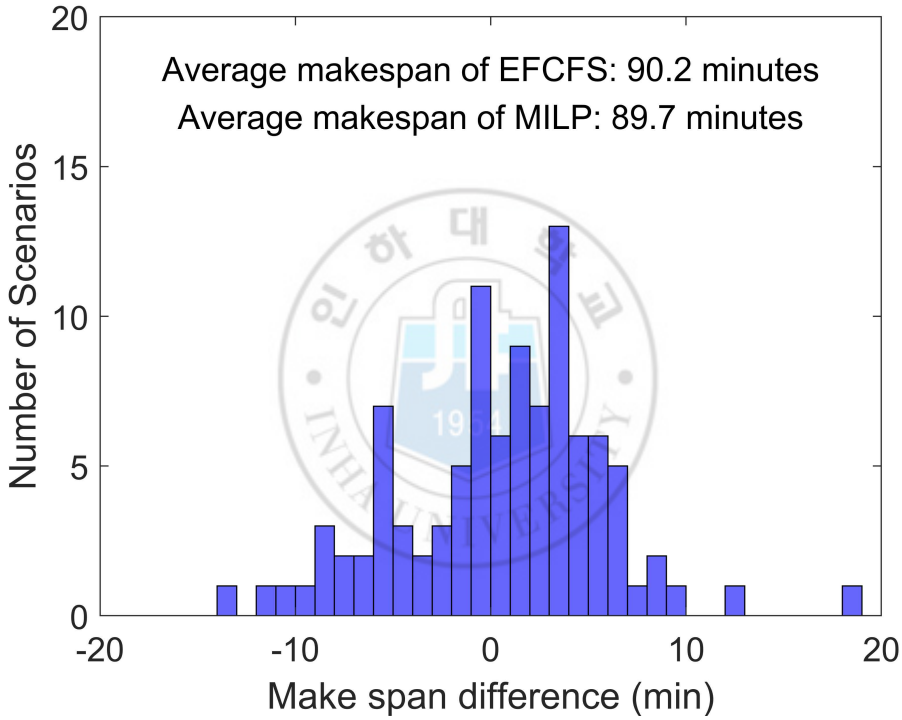
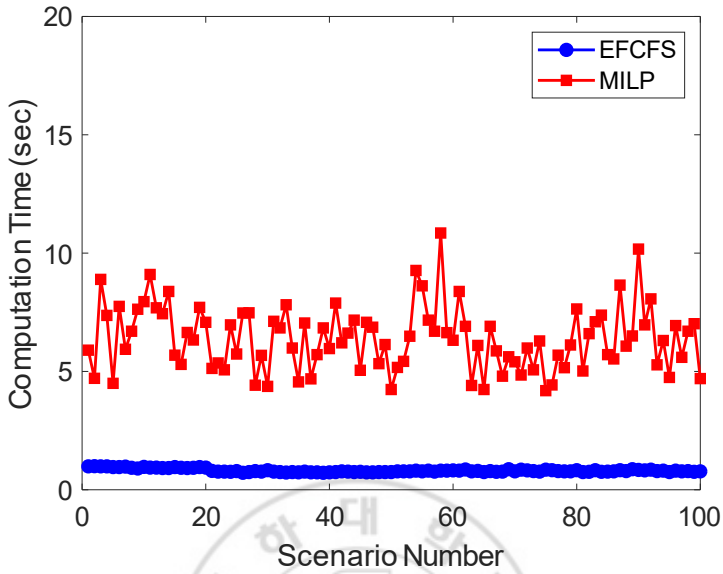
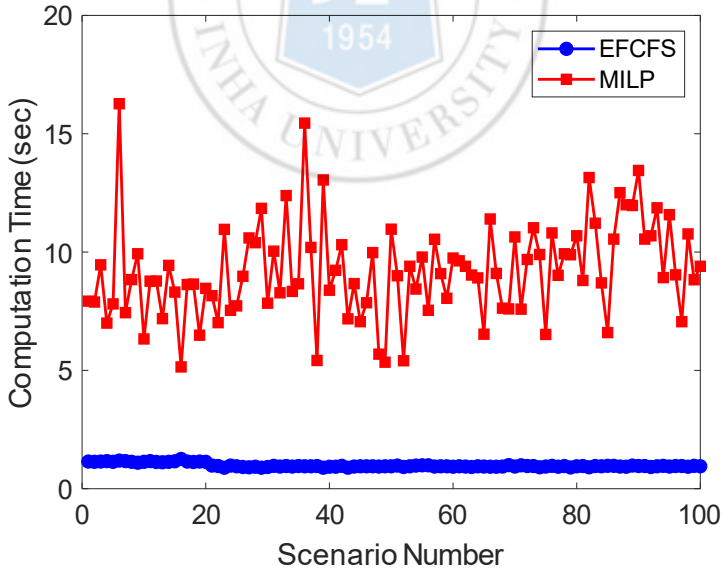


그림 4.22 100개 시나리오에 대한 EFCFS와 MILP 스케줄러의 makespan 차이



(a)



(b)

그림 4.23 EFCFS와 MILP 스케줄러의 계산 시간 비교

5. 수정 스케줄링

출도착 스케줄러는 초 단위의 계산을 통해 정밀히 조정된 스케줄을 제공한다. 그러나 실제 환경에서 항공기들을 제공받은 스케줄대로 운용하는 것은 불가능하며, 아무리 스케줄러가 모든 조건을 고려했더라도 실제 운용에서는 다양한 오차가 발생할 수 있다.

수정 스케줄링 알고리즘은 스케줄러와 실제 운용 환경을 연동하여 이러한 오차를 줄이고, 항공기 운용에 충분히 적용 가능한 스케줄링 결과를 제공할 수 있도록 한다. 수정 스케줄링 알고리즘의 구조는 그림 5.1과 같다. 먼저 운용자의 요구조건-우선순위, 속도 제한 등-을 반영한 스케줄러가 조정된 스케줄을 제공하면, 항공기들은 이를 기반으로 주어진 명령을 이행한다. 실제 운용 환경에서는 항공기 간 충돌 감지 및 회피(Conflict Detection and Resolution, CD&R)를 수행하는 등 항공기 운용에 이상이 없는지 확인한다. 일정 시간 이후, 움직이고 있는 항공기들의 데이터를 기반으로 다시 스케줄링을 수행한다. 항공기들은 현재 시점을 기준으로 업데이트된 결과를 제공받아 운용된다.

수정 스케줄링은 스케줄러가 실시간에 가까운 속도로 스케줄을 제공해야 그 효율이 증가하며, EFCFS 스케줄러는 계산 속도가 빠르고 검증된 성능을 가지고 있으므로 수정 스케줄링에 적합하다.

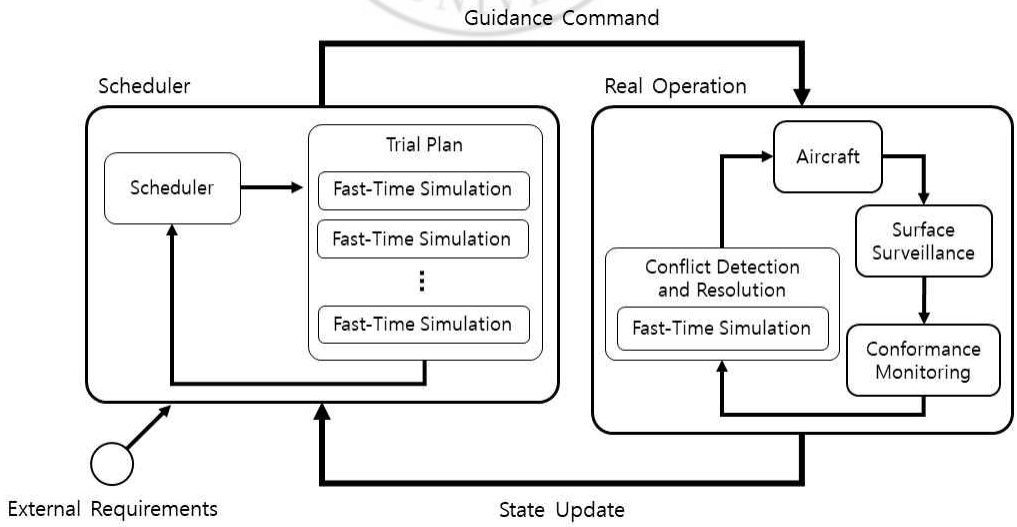


그림 5.1 수정 스케줄링 알고리즘

본 장은 EFCFS 스케줄링 알고리즘을 기반으로 한 '수정 EFCFS(Corrective EFCFS, C-EFCFS)' 스케줄링 기법을 다룬다. 먼저 스케줄링 결과 검증과 실제 운용 환경 모사를 위한 배속 시뮬레이션에 대해 설명한다. 또한, 스케줄러에서 계산한 스케줄과 실제 운용 데이터를 기반으로 재스케줄링을 수행하는 과정을 상세히 서술하고, 실제와 유사한 시나리오를 활용한 스케줄링 결과를 분석한다.

5.1. 공항 배속 시뮬레이션

공항 배속 시뮬레이션(Fast-Time simulation) 도구는 EFCFS 스케줄러에서 계산된 스케줄링 결과를 입력받아 실제 지상 운용을 모사한다. 그림 5.2는 배속 시뮬레이션 도구로 지상 운용을 모사하는 것을 보여준다. 배속 시뮬레이션 도구에 적용된 항공기 운동 모델은 1차원 및 2차원 질점 운동 모델이며, 2차원 모델의 입력 변수 및 상태 변수는 표 5.1과 같다. 제어기의 경우, 항공기가 경로상의 노드를 순차적으로 통과하기 위한 방위각 제어기와 주어진 스케줄을 준수하기 위한 속도제어기로 구성되어 있다 [54].

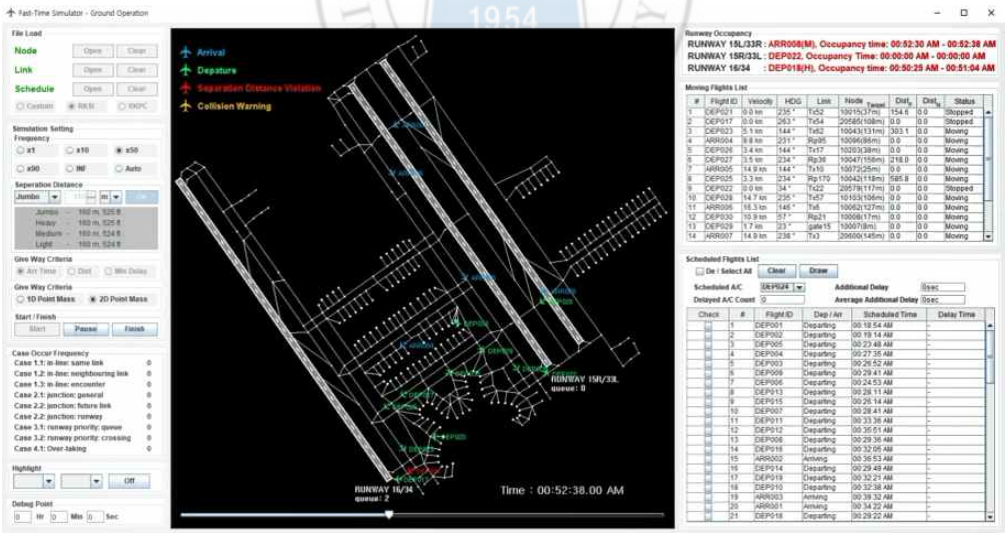


그림 5.2 배속 시뮬레이션 도구 [54]

표 5.1 항공기 2차원 질점 운동 모델 변수

Type	Variables
Input commands	Nose gear steering angle, δ
	Engine thrust, T
States	Position, (x, y)
	Heading, Ψ
	Ground speed, v
	Weight, W

수정 스케줄링 기법에서 공항 배속 시뮬레이션은 크게 3가지의 역할을 수행한다. 첫 번째는 스케줄링 결과를 검증하는 것으로, 스케줄러에서 제공받은 스케줄대로 실제 항공기 운용이 가능한지 확인한다. 두 번째 역할은 스케줄링 결과를 실제 운용에 적용하기 전에 배속 시뮬레이션 도구에서 CD&R 및 활주로 분리 등을 통해 스케줄을 미리 조정하는 것으로, 이를 통해 실제 운용 시 발생할 수 있는 위험 상황을 줄일 수 있다. 마지막은 실제 운용 환경을 대신하기 위한 대체 요소(Surrogate)의 역할이다.

본 논문에서는 실제 운용 환경을 배속 시뮬레이션으로 대체하였으며, 그림 5.1의 구조를 그림 5.3과 같이 단순화하여 수정 스케줄링을 수행하였다.

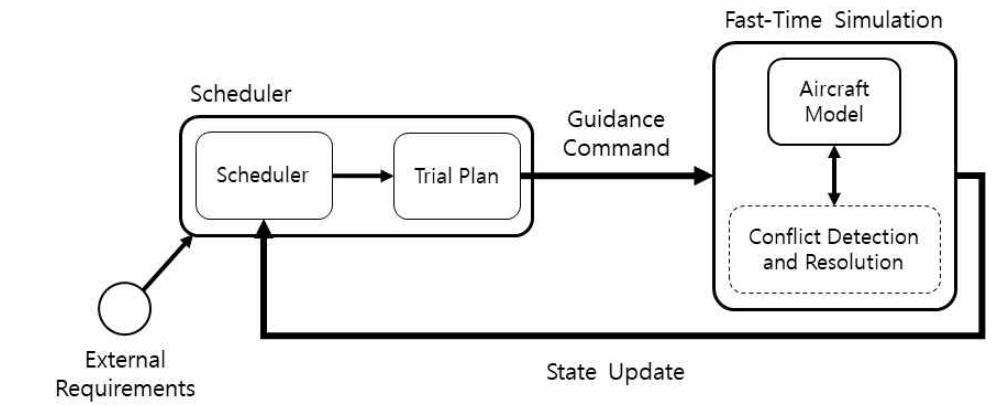


그림 5.3 단순화된 수정 스케줄링 알고리즘

5.2. 수정 스케줄링 알고리즘

수정 스케줄링 알고리즘은 '스케줄링 - 항공기 운용 - 상태 업데이트(스케줄 재구성)'의 과정을 일정 주기(Cycle)로 반복한다. 그림 5.4는 이 과정을 시간 순으로 배열한 것이다. T_i, T_f, t_w는 각각 수정 스케줄링 시작 시간, 종료 시간, 주기를 의미하며, t_i는 각 주기의 시작 시점, t_f는 각 주기의 종료 시점을 의미한다. 맨 처음 주기는 t_{i1}을 기준으로 스케줄링과 실제 운용을 수행한다. 스케줄링은 t_{i1}부터 종료 시간인 T_f까지 전체 구간을 대상으로 하며, 운용은 t_{i1}부터 t_w 동안 수행한다. 운용이 종료된 직후, t_{f1}을 기준으로 스케줄을 재구성하며, 재구성된 스케줄은 다음 주기의 스케줄링에 입력된다. 이 때, t_{i1}은 스케줄링 시작 시간 T_i와 동일하며, 다음 주기의 시작 시점 t_{i2}는 이전 주기의 종료 시점 t_{f1}과 동일하다. T_f는 스케줄러에서 계산되는 항공기 지연 시간에 따라 증가할 수 있다.

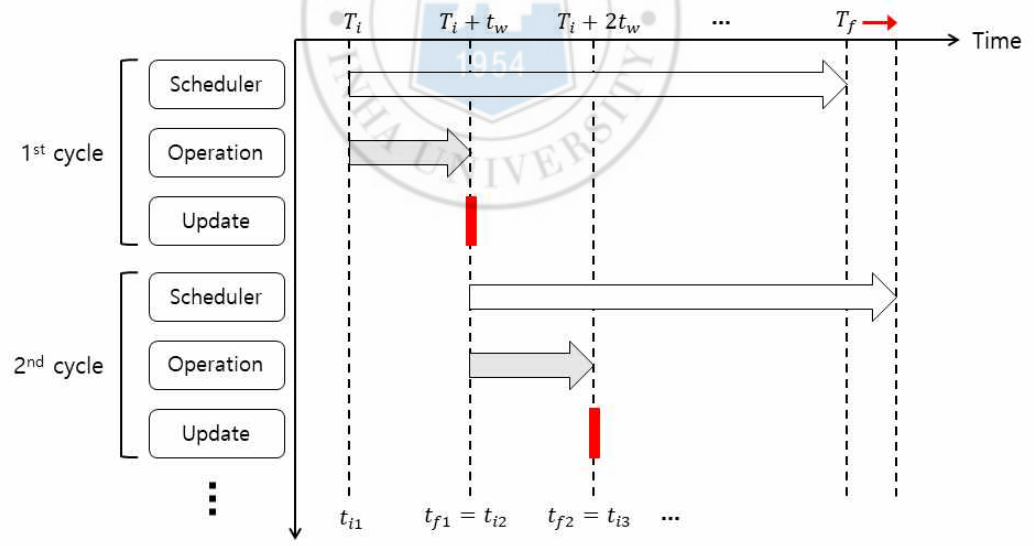


그림 5.4 스케줄 시간에 따른 수정 스케줄링 과정

매 주기마다 반복되는 과정 중 가장 마지막에 수행되는 상태 업데이트는 스케줄링 우

선순위에 관계없이 각 항공기마다 동시 수행이 가능하기 때문에, 프로그래밍 시 '멀티스레드(Multi-thread)'를 활용한 '병렬 프로세스(Parallel processing)' 구현이 가능하다. 상태 업데이트는 '노드-링크 분할 - 경로 재생성 - 상태 업데이트 - 스케줄 재구성' 과정으로 구성되며, 본 절은 각 세부 과정에 대해 설명한다.

5.2.1. 노드-링크 분할

매 주기마다 항공기 운용이 종료되면, 종료 시점에서의 항공기들의 위치를 기준으로 노드-링크 모델을 분할한다. 이때, 불필요한 계산을 줄이기 위해 종료 시점에 노드-링크 위에 존재하는 항공기들만 고려한다.

그림 5.5는 노드-링크 분할 과정을 보여준다. 항공기와 가장 가까운 링크를 항공기가 현재 지나가는 링크로 간주하며, 항공기 위치와 해당 링크의 수선의 발을 기준으로 링크를 분할한다. 이때 수선의 발이 새로운 가상 노드(Virtual node)로 정의되며, 분할된 링크들은 이 가상 노드를 공유한다. 그림 5.6은 시뮬레이션 종료 후 분할된 노드-링크의 예시이다.

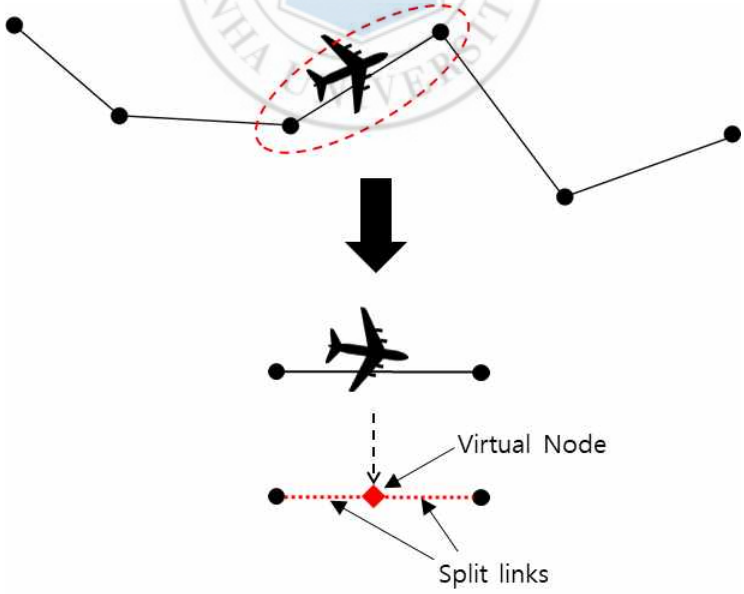


그림 5.5 항공기 위치 기준 노드-링크 분할

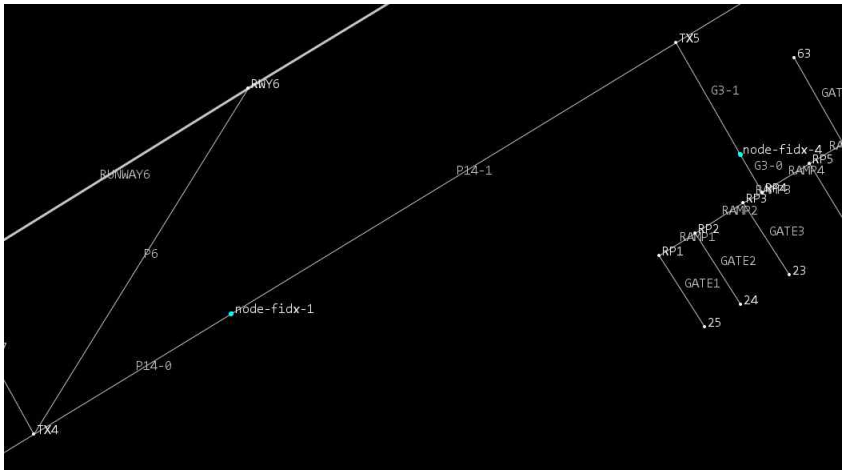


그림 5.6 분할된 노드-링크 예시

5.2.2. 항공기 경로 재생성

노드-링크 분할이 완료되면 종료 시점에서의 항공기들의 위치로부터 목적지까지의 항공기 이동 경로를 재생성한다. 이는 앞서 설명한 경로 할당을 수행하기 위함이며, 매 주기마다 스케줄링 환경이 변하므로 경로 할당의 결과도 매 주기마다 다를 수 있다.

그림 5.7, 그림 5.8은 각각 출발 항공기와 도착 항공기의 최대 경로 탐색 범위를 보여준다. 항공기의 이착륙 경로는 고정되어 있으므로, 출발 항공기는 현재 위치로부터 활주로 시단까지, 도착 항공기는 활주로 탈출 노드부터 게이트까지의 범위를 탐색한다.

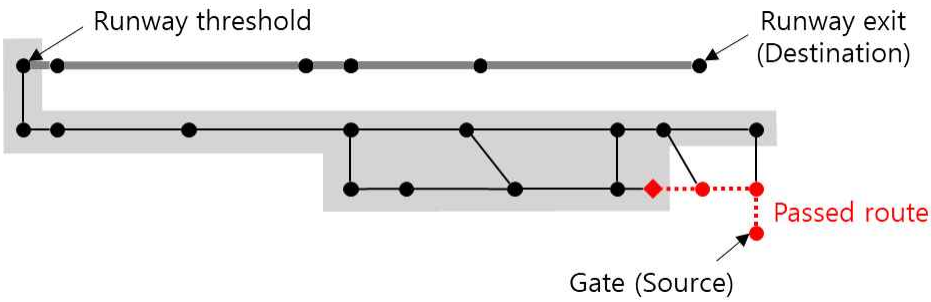


그림 5.7 출발 항공기의 최대 경로 탐색 범위

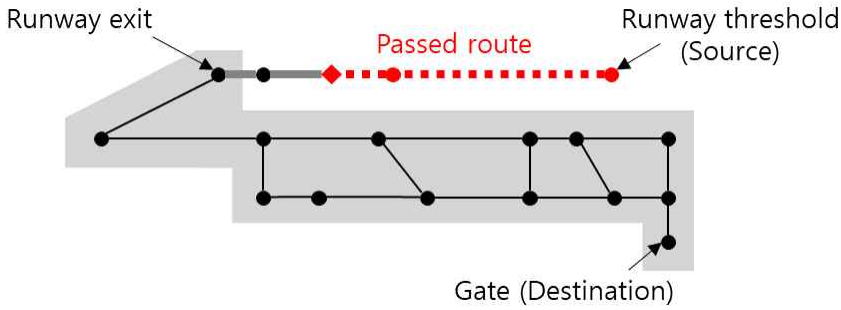


그림 5.8 도착 항공기의 최대 경로 탐색 범위

5.2.3. 상태 업데이트 및 스케줄 재구성

노드-링크 분할과 경로 재생성이 완료되면, 항공기 궤적을 기반으로 기존 스케줄링 결과와 노드-링크를 업데이트하여 실제 운용으로 인한 오차를 반영한다.

그림 5.9는 상태 업데이트 및 스케줄 재구성을 보여준다. 먼저, 모든 항공기의 스케줄 경로를 분할된 노드-링크 모델로 대체하고 종료 시점까지의 항공기 궤적은 스케줄링 결과로 치환한다. 이후 치환된 스케줄링 결과를 이용해 분할된 노드-링크 모델을 업데이트한다. 실제 항공기 궤적은 그대로 다음 주기의 노드-링크 모델 제약 조건으로 적용된다. 현재 위치로부터 도착 노드까지 재생성된 경로는 다음 주기에 입력될 스케줄로 재구성되며, 이때 스케줄 시작 시간은 종료 시점의 시간으로 적용된다.

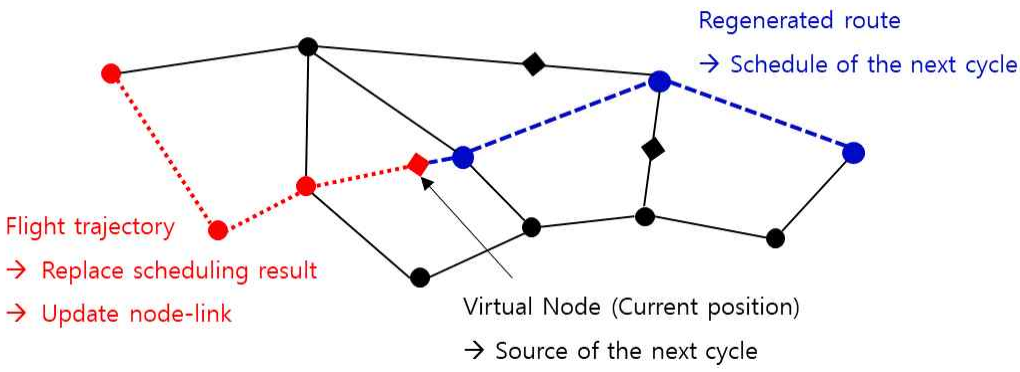


그림 5.9 분할된 노드-링크에서의 상태 업데이트 및 스케줄 재구성

그림 5.10은 스케줄러가 계산한 스케줄링 결과 예시이며, 그림 5.11은 이를 기반으로 항공기를 운용한 실제 궤적을 시간-거리 평면상에 나타낸 것이다. 분할된 노드-링크 모델은 그림 5.12와 같이 기존 스케줄의 이동 경로에 그대로 반영된다. 그림 5.13 및 그림 5.14는 스케줄 업데이트 및 재구성 과정으로, 재생성된 경로가 기존 경로와 동일한 경우의 예시이다. 이때 가상 노드는 다음 주기의 스케줄링 시작 지점이므로 제약 조건을 적용하지 않는다. 상태 업데이트가 완료된 이후 상세 propagation 과정은 부록 B에 정리하였다.

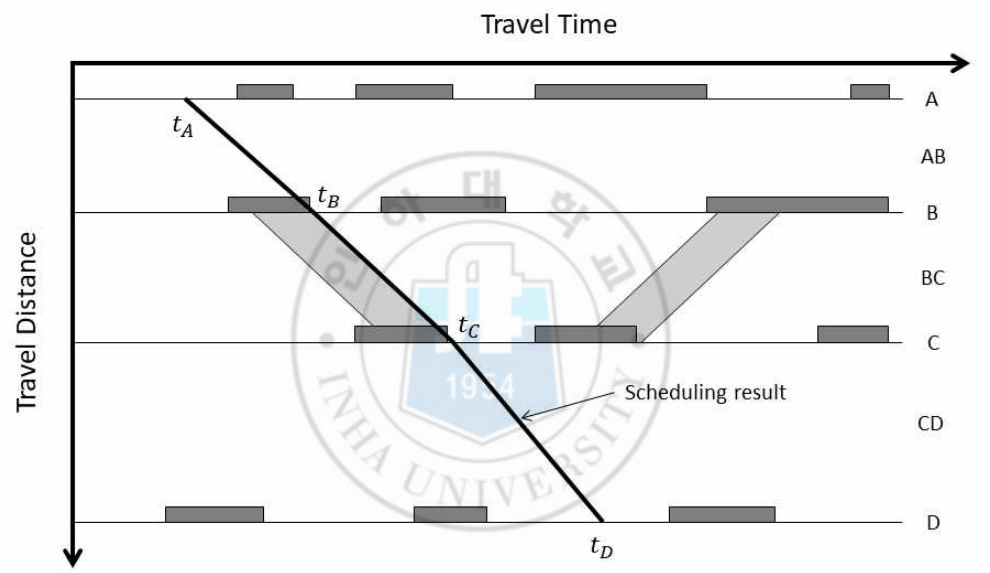


그림 5.10 한 주기(Cycle)의 스케줄링 결과

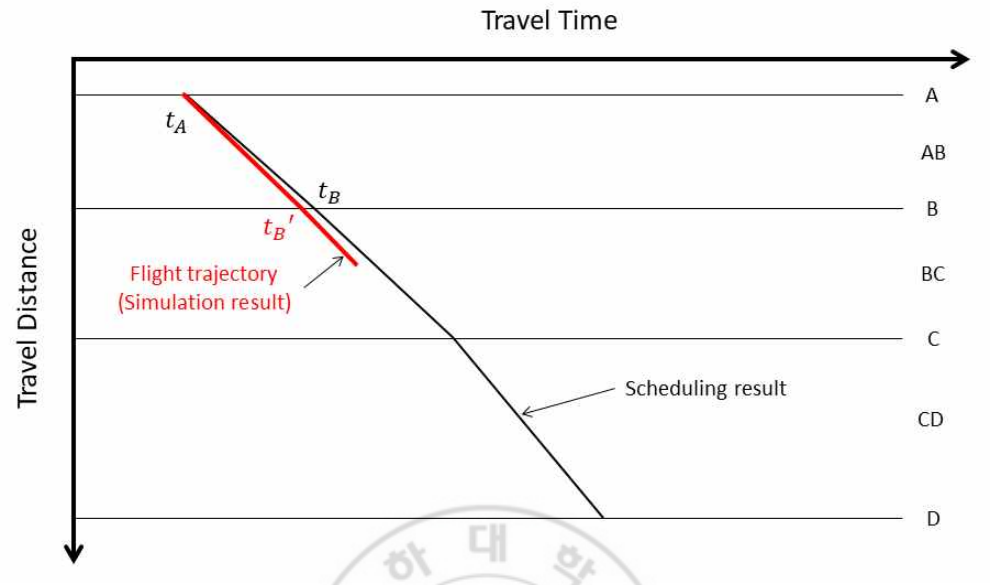


그림 5.11 스케줄링 결과 및 실제 운용 궤적

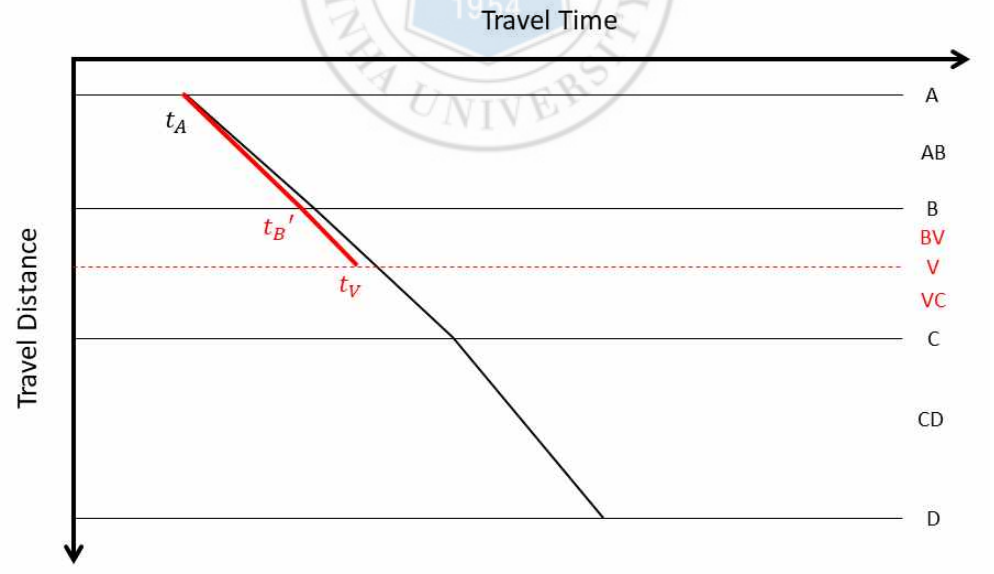


그림 5.12 분할된 노드-링크 모델로 대체된 이동 경로

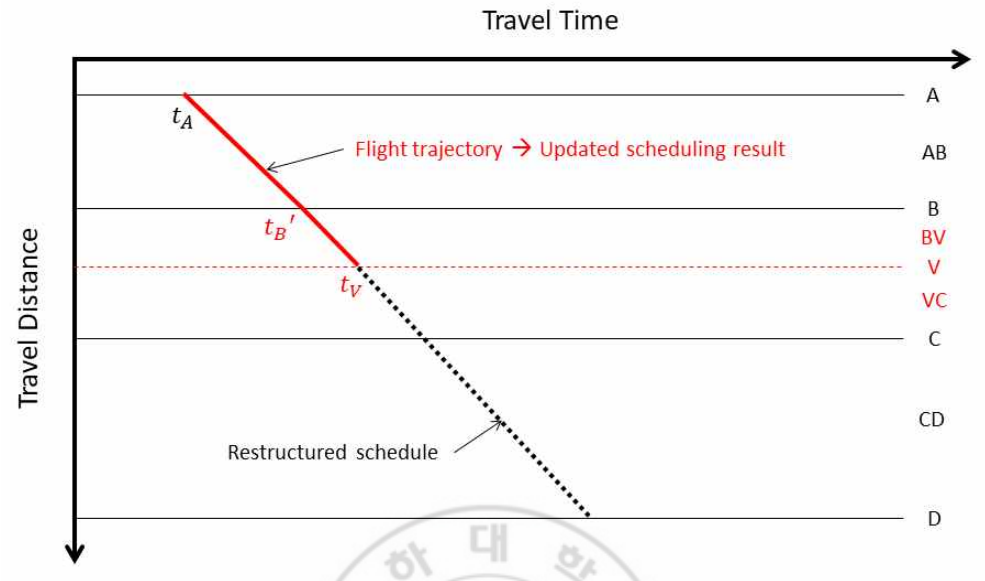


그림 5.13 항공기 궤적으로 대체된 스케줄링 결과 및 재구성된 스케줄

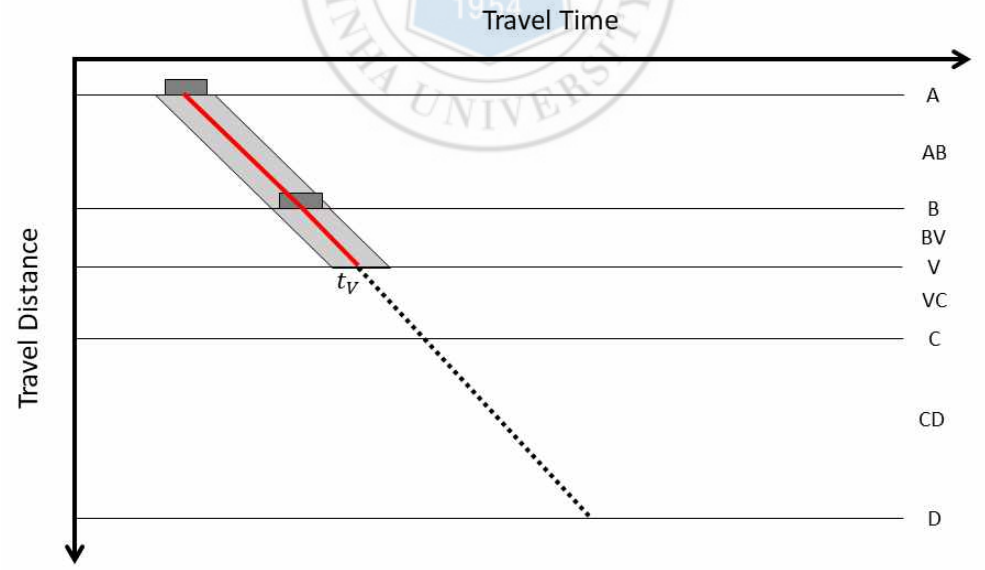


그림 5.14 실제 궤적이 다음 주기의 노드-링크 제약 조건으로 적용된 모습

5.3. 스케줄링 결과

EFCFS 스케줄러와 C-EFCFS 스케줄러의 스케줄링 결과를 비교하였으며, 각 스케줄러에 대해 고정된 경로만을 고려한 스케줄링과 경로 할당을 적용한 스케줄링을 수행하여 그 결과를 비교하였다. 경로 할당은 최대 3개의 후보 경로를 탐색하도록 하였다.

스케줄링 시나리오는 인천 국제공항에서 1시간 동안 총 63대의 항공기가 출도착하는 상황을 가정해 구성하였다. 출발 항공기는 35대, 도착 항공기는 28대이며, 이는 2019년 8월 월평균 출도착 데이터 중 오전 10시부터 11시 사이에 기록된 것을 기반으로 한다. 이때 스케줄링 전후의 교통 상황은 고려하지 않았다.

항공기들의 SOBT 및 SLDT는 오전 10시 ~ 11시 사이의 시간을 임의로 지정하였으며, 출도착 게이트는 제 1, 2 터미널에 임의 배정하였다. 출발 항공기의 이동 경로는 게이트에서 활주로 시단까지 자동 생성되며, 활주로 이륙 경로는 시단에서 이륙 거리만큼 생성된다. 도착 항공기의 경우, 활주로 시단에서 활주로 탈출 노드까지 착륙 경로를 생성한 뒤 탈출 노드부터 도착 게이트까지 지상 이동 경로를 생성한다. 활주로 분리 기준은 부록 A에 정의된 기준을 적용하였다.

표 5.2는 항공기들의 구성과 사용하는 활주로를 정리한 것이다. 출발 항공기는 활주로 34와 33L에서 3:2 비율로 이륙하며, 도착 항공기는 활주로 33R에서만 착륙을 수행한다. Heavy 및 Medium 등급의 비율은 약 2:1로 구성하였다.

표 5.2 출도착 활주로 및 항공기 구성

	Runway	Heavy	Medium	Total
Departure	34	9	12	21
	33L	14	0	14
Arrival	33R	18	10	28
Total		41	22	63

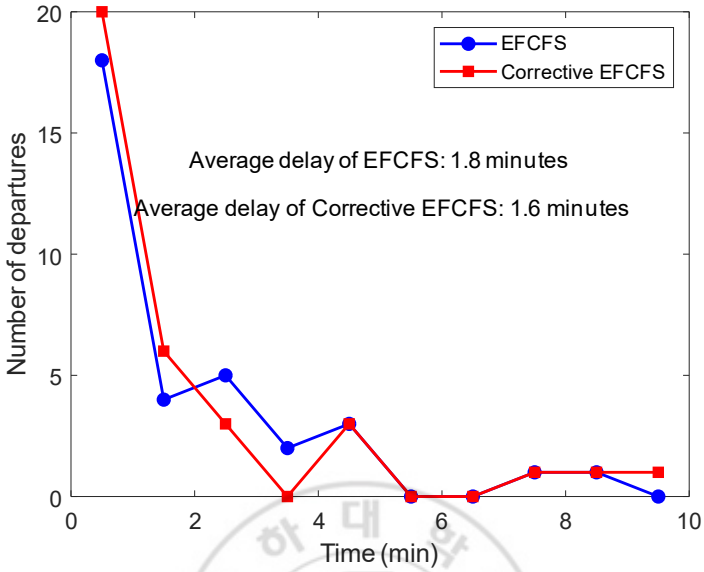
스케줄링 우선순위는 SOBT 및 SLDT 순서로 설정하였다. 공항에서의 지상 이동 속도는 최대 10%까지 감속하도록 제한하였으며, Link blocking time은 20초로 설정하였

다. C-EFCFS 스케줄러의 주기는 10분으로 설정하였으며, 배속 시뮬레이션 수행 시 항공기 운동 모델은 주어진 스케줄 명령보다 빠르게 움직인다. 또한 시뮬레이션 과정에 활주로 분리 조건 및 CD&R은 적용하지 않았다.

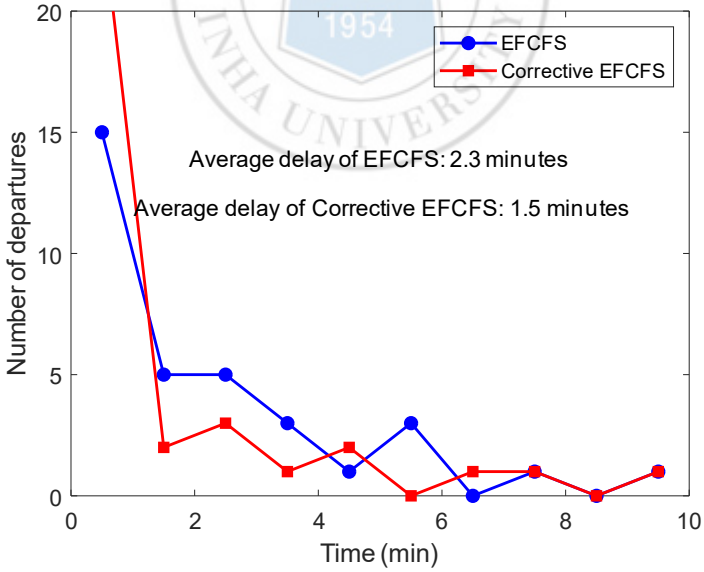
EFCFS 및 C-EFCFS 스케줄러는 범용성 및 개발의 편리성을 위해 자바(Java 1.8) 기반으로 개발되었으며, 항공기의 스케줄 경로 생성 및 C-EFCFS 스케줄러의 상태 업데이트 과정은 병렬 처리를 구현하여 빠른 계산이 가능하도록 하였다. 스케줄링은 AMD Ryzen™ Threadripper™ 2950X (16 cores), RAM 64GB, Window 10 환경에서 수행되었다.

그림 5.15, 5.16은 고정 경로에 대한 EFCFS 스케줄러와 C-EFCFS 스케줄러의 항공기 지연 시간 분포를 비교한 것이다. 그림 5.15 (a)는 출발 항공기의 DOBT 분포이며, EFCFS와 C-EFCFS의 평균 DOBT는 각각 1.8분, 1.6분이다. 그림 5.15 (b)는 DTOT 분포를 보여준다. EFCFS와 C-EFCFS의 평균 DTOT는 각각 2.3분, 1.5분으로 EFCFS의 항공기 지연 시간이 크게 증가하였다. 1분 ~ 4분 구간은 EFCFS가 C-EFCFS보다 많이 분포해 있으나, C-EFCFS는 1분 이하 구간에 이륙 지연이 집중되어 있다. 이를 통해, C-EFCFS의 경우 주어진 스케줄보다 빨리 이동하는 항공기 운동 모델로 인해 택시 지연이 발생하지 않아 평균적으로 더 일찍 이륙하였음을 알 수 있다.

도착 항공기의 지연 시간 분포는 출발 항공기와 거의 유사한 변화 양상을 보인다. 그림 5.16 (a)는 도착 항공기의 DLDT 분포이다. EFCFS의 평균 DLDT는 6.5분이며, C-EFCFS의 평균 DLDT은 7.4분으로 C-EFCFS가 EFCFS보다 더 큰 지연을 보인다. 그림 5.16 (b)는 DIBT 분포이다. EFCFS의 평균 DIBT는 활주로 착륙 지연과 동일하며, 그 분포 또한 큰 변화가 없다. 반면, C-EFCFS의 평균 DIBT는 6.9분으로 항공기들이 주어진 명령보다 빨리 게이트에 도착한 것을 확인할 수 있다.

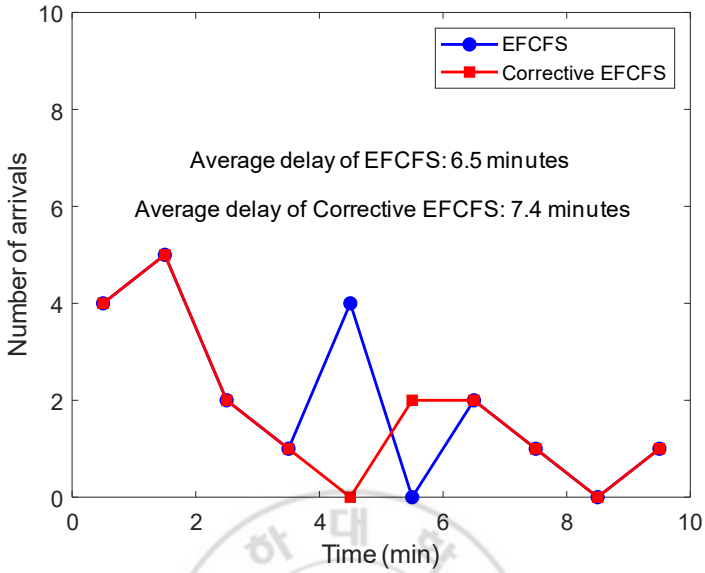


(a) DOBT

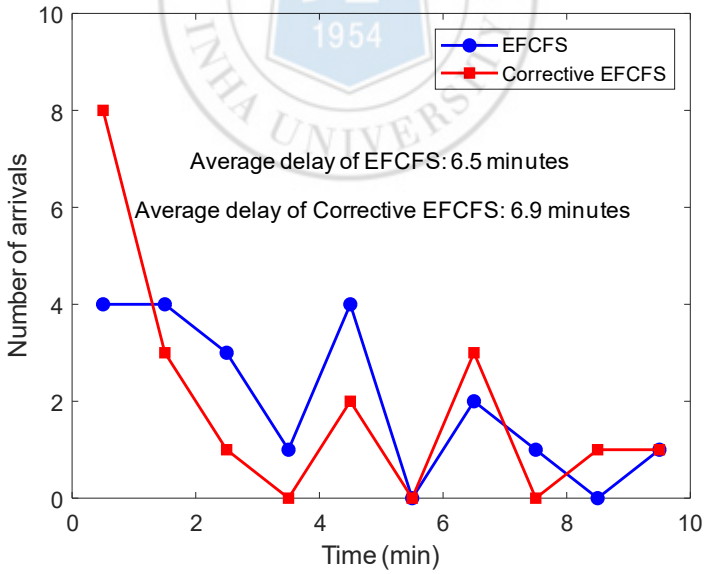


(b) DTOT

그림 5.15 출발 항공기 지연 시간 분포



(a) DLDT



(b) DIBT

그림 5.16 도착 항공기 지연 시간 분포

그림 5.17 및 그림 5.18은 경로 할당을 적용하여 스케줄링을 수행한 결과를 비교한 것이다. 그림 5.17은 EFCFS와 C-EFCFS에서 계산한 출발 항공기의 지연 시간 분포이다. 그림 5.17 (a)는 DOBT 분포로, EFCFS와 C-EFCFS의 평균 DOBT은 각각 1.7분, 1.9분이다. EFCFS의 평균 지연 시간은 소폭 감소하였으나, C-EFCFS의 경우 지연 시간이 소폭 증가하였다. 그림 5.17 (b)는 DTOT 분포를 보여준다. 1분 ~ 4분 구간은 EFCFS가 더 많이 분포하나, 1분 이하 구간의 경우 C-EFCFS가 더 많이 집중되어 있다. 평균 DTOT는 EFCFS가 2.2분, C-EFCFS는 2.3분으로 C-EFCFS의 평균 택시 지연이 더 적게 발생하였다.

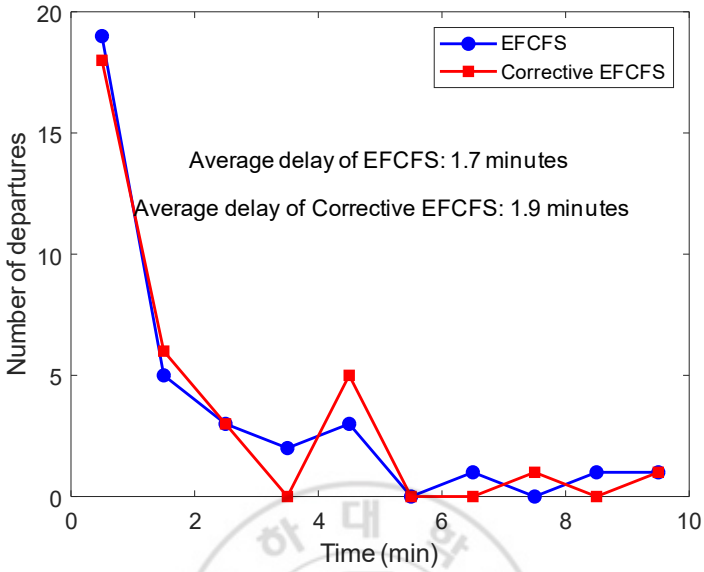
그림 5.18 (a)는 도착 항공기의 DLDT 분포를 보여준다. EFCFS와 C-EFCFS의 평균 DLDT는 각각 6.6분, 7,4분으로 C-EFCFS의 지연 시간이 더 크게 발생하였다. 그림 5.18 (b)는 DIBT 분포이다. EFCFS의 평균 DIBT는 변화가 없으나, C-EFCFS의 평균 DIBT는 6.9분으로 크게 감소하였다.

표 5.3은 스케줄러에서 계산된 평균 지연 시간을 정리한 것이다. 고정 경로 스케줄링 결과와 비교했을 때, C-EFCFS에서 계산된 출발 항공기의 평균 지연 시간은 오히려 증가하였으나, 항공기 운동 모델의 특성으로 인해 택시 지연은 비교적 적게 발생하였음을 확인하였다.

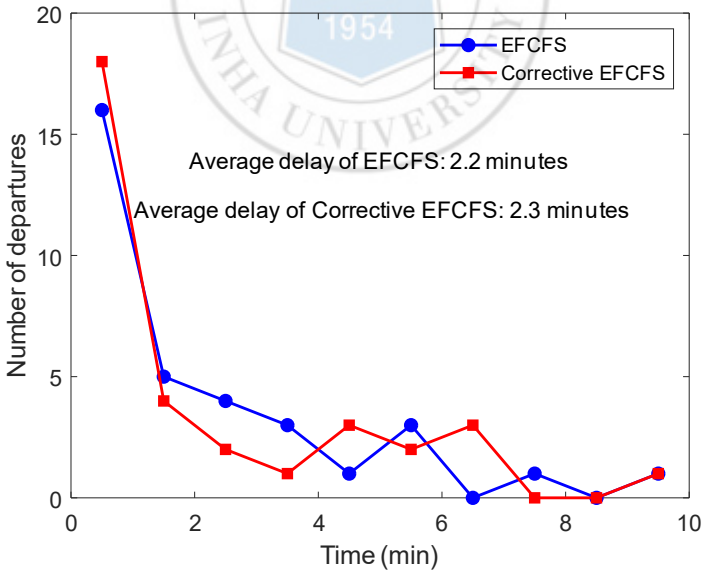
실제와 유사한 상황의 시나리오를 구성하여 스케줄링 결과를 분석하였으나, EFCFS에 비해 C-EFCFS는 뚜렷한 경향성을 보이지 않았다. 수정 스케줄링 기법은 항공기들의 움직임 및 반복 스케줄링으로 인한 불확실성이 존재한다. 따라서 향후 몬테카를로 시뮬레이션과 다양한 성능 지표를 활용한 상세 분석이 요구된다.

표 5.3 스케줄러에서 계산된 평균 지연 시간 (단위: 분)

	Route assignment	DOBT	DTOT	DIBT	DLDT
EFCFS	Fixed	1.8	2.3	6.5	6.5
	3 Routes	1.7	2.2	6.6	6.6
C-EFCFS	Fixed	1.6	1.5	6.9	7.4
	3 Routes	1.9	2.3	6.9	7.4

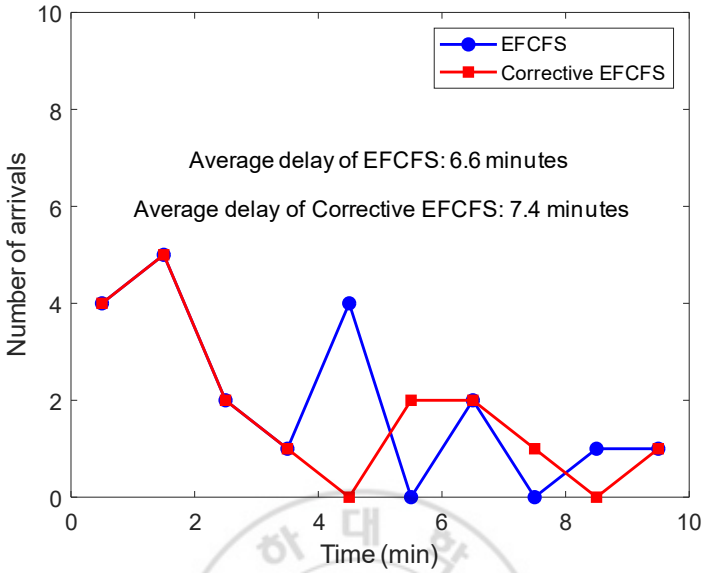


(a) DOBT

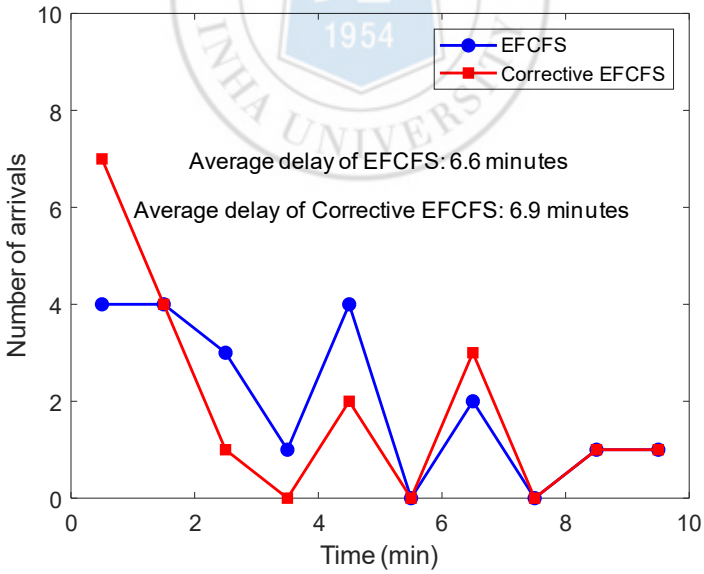


(b) DTOT

그림 5.17 출발 항공기 지연 시간 분포



(a) DLDT



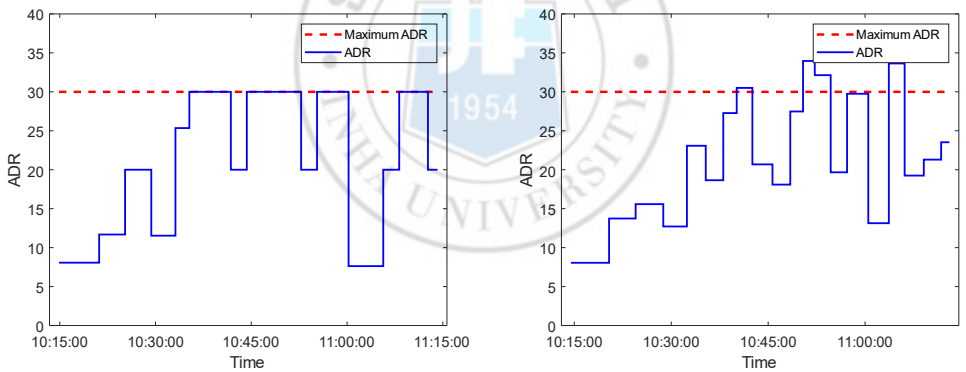
(b) DIBT

그림 5.18 도착 항공기 지연 시간 분포

그림 5.19는 경로 할당을 적용하여 스케줄링을 수행하였을 때 인천 국제공항의 활주로 34 이용률 변화를 그래프로 나타낸 것이다. 그림 5.19 (a)는 EFCFS의 활주로 이용률이며, 활주로 분리 기준으로 인해 ADR 제약 조건이 제대로 적용된 것을 확인할 수 있다. 그림 5.19 (b)는 C-EFCFS의 활주로 이용률을 보여준다. 이는 배속 시뮬레이션 과정에는 활주로 분리 기준이 적용되지 않아 항공기 운동 모델이 스케줄러에서 처리한 제약 조건을 위반했기 때문이다. 시뮬레이션 과정에 활주로 분리 기준을 적용할 경우, C-EFCFS 역시 스케줄러의 제약 조건에 위배되지 않는 결과를 제공한다.

표 5.4는 각 스케줄러의 계산 시간을 측정한 것이다. EFCFS는 경로 할당을 수행할 경우 스케줄링 계산 시간이 후보 경로 수에 비례하여 증가하였다. 그러나 C-EFCFS의 경우 재스케줄링을 반복하는 과정으로 인해 계산 시간이 불규칙하게 증가하였다.

그림 5.20은 C-EFCFS 스케줄러가 스케줄링을 수행할 때의 CPU 사용량으로, 상태 업데이트 과정에서 CPU의 모든 코어를 사용하는 것을 확인하였다.



(a) EFCFS (b) C-EFCFS

그림 5.19 인천 국제공항 활주로 34 이용률

표 5.4 스케줄러 계산 시간 (단위: 초)

	Fixed route	Route assignment
EFCFS	9.6	26
C-EFCFS	151.4	236.4

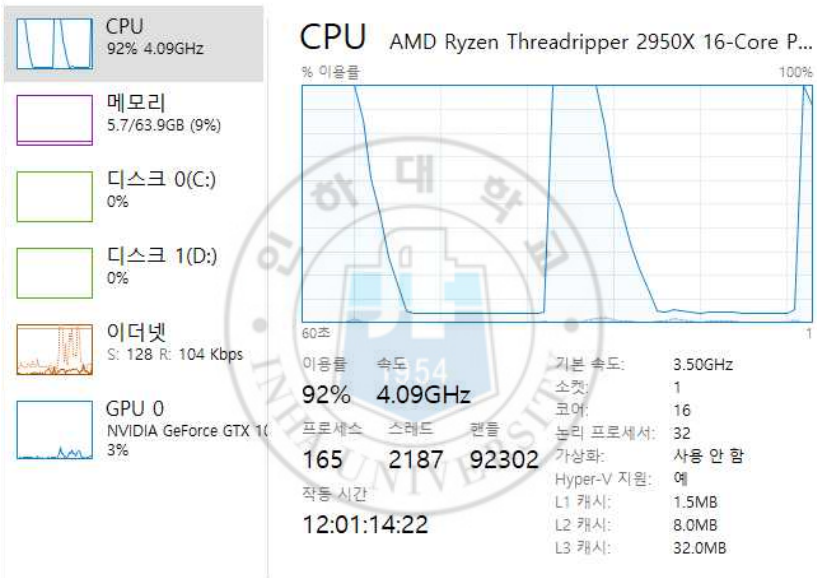


그림 5.20 C-EFCFS 스케줄러의 CPU 사용량 추이

6. 결론

6.1. 결론 및 요약

본 논문은 공항 지상 운용을 위한 FCFS 기반 스케줄링 기법을 제시하였다. 제시된 기법은 일반적인 FCFS 알고리즘의 효율성을 증대하고 성능을 개선한 것으로, 기존 기법에 다양한 제약 조건을 반영하고, 주어진 순서대로 스케줄링하지 않고 순서에 변화를 주는 것을 가능하도록 하였다. 단순한 구조와 빠른 계산 속도를 유지하면서 주어진 제약 조건을 모두 만족하는 결과를 도출할 수 있으며, 다양한 이동 경로와 실제 운용 환경을 고려할 시 보다 유연하고 높은 충실도의 스케줄링이 가능하다. 국내 항적 데이터를 기반으로 공역 스케줄링을 수행하였으며, 인천 국제공항을 대상으로 스케줄링을 수행하였다. 또한, 최적화 기반 스케줄링 알고리즘과 그 결과를 비교하여 본 연구에서 제시한 기법이 거의 비슷한 성능을 가지고 있음을 확인하였다.

수정 스케줄링 기법은 일정 주기마다 실제 항공기 상태를 기반으로 재스케줄링을 수행하여 실제 운용 환경에 적합한 결과를 제시한다. 본 논문은 배속 시뮬레이션을 활용하여 실제 운용 환경을 모사하였다. 배속 시뮬레이션은 스케줄링 결과 검증, 항적 모니터링을 통한 스케줄 조정 등 다양한 역할을 수행하여 스케줄러가 높은 충실도의 결과를 제시할 수 있도록 한다. 재스케줄링으로 인해 기존보다 계산 시간이 증가하지만, 이는 적절한 병렬 처리 구조를 통해 해결할 수 있다.

6.2. 향후 계획

본 연구에서 제시한 기법을 실제 운용 환경에 적용하기 위해서는 충분한 성능 검증이 요구되며, 특히 수정 스케줄링에서 경로 할당의 효과는 보다 상세한 분석이 필요하다.

따라서 향후 실제 항공기와 유사한 수준의 운동 모델을 적용하고, 다양한 성능 지표를 구성하여 실제 데이터 기반의 스케줄링 결과를 분석할 계획이다. 또한, 스케줄러의 성능을 향상시킬 수 있는 최적의 스케줄링 우선순위 설정 방법을 제시할 예정이다.

부록 A. 인천 국제공항 활주로 분리 기준

인천 국제공항은 2020년 6월 기준 총 3개의 활주로(15L/33R, 15R/33L, 16/34)를 운용하고 있으며, 이중 15L/33R과 15R/33L은 '근접 평행(Closed parallel)' 활주로이다. 15L/33R은 도착 항공기 전용, 15R/33L과 16-34는 출발 항공기 전용으로 운용되고 있다. 3개의 활주로는 항상 같은 방향(Designation)으로 운용되기 때문에 활주로 반대편에서 이착륙할 때의 활주로 분리 기준은 정의되지 않는다.

본 연구에서 정의한 인천 국제공항의 활주로 분리 기준은 다음과 같다. 표 A.1~A.4는 동일 활주로에서 운용되는 선행-후행 항공기의 최소 분리 시간을 정리한 것이며, 표 A.5는 인접 활주로에서 운용되는 선행-후행 항공기의 최소 분리 시간을 정리한 것이다. 인접 활주로에서 선행 항공기가 도착한 뒤 후행 항공기가 출발할 때의 최소 분리 시간은 모두 0초로 정의되었다. 또한, 15R/33L은 출발 전용, 15L/33R은 도착 전용이기 때문에, 인접 활주로에서 선행 항공기가 출발한 뒤 후행 항공기가 출발하거나 선행 항공기가 도착한 뒤 후행 항공기가 도착하는 경우는 정의되지 않는다.

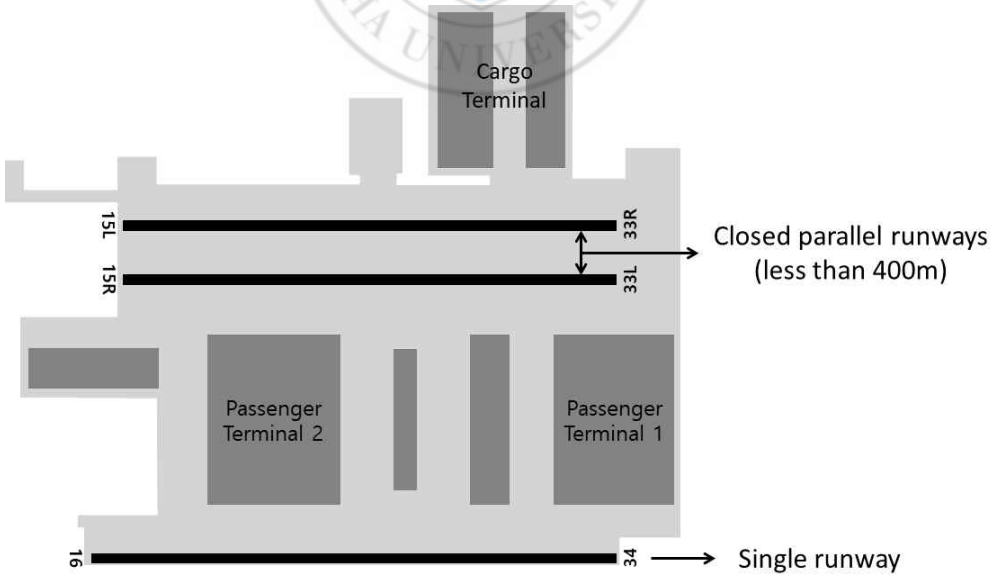


그림 A.1. 인천 국제공항의 활주로 배치

표 A.1. 동일 활주로에서의 '선행 출발 – 후행 출발' 최소 분리 시간 (단위: 초)

		Trail			
		Light	Medium	Heavy	Jumbo
Lead	Light	120	120	120	120
	Medium	180	120	120	120
	Heavy	180	180	120	120
	Jumbo	180	180	120	120

표 A.2. 동일 활주로에서의 '선행 출발 – 후행 도착' 최소 분리 시간 (단위: 초)

		Trail			
		Light	Medium	Heavy	Jumbo
Lead	Light	120	120	120	120
	Medium	120	120	120	120
	Heavy	120	120	120	120
	Jumbo	180	180	120	120

표 A.3. 동일 활주로에서의 '선행 도착 – 후행 출발' 최소 분리 시간 (단위: 초)

		Trail			
		Light	Medium	Heavy	Jumbo
Lead	Light	120	120	120	120
	Medium	120	120	120	120
	Heavy	120	120	120	120
	Jumbo	180	180	120	120

표 A.4. 동일 활주로에서의 '선행 도착 – 후행 도착' 최소 분리 시간 (단위: 초)

		Trail			
		Light	Medium	Heavy	Jumbo
Lead	Light	180	120	120	120
	Medium	180	120	120	120
	Heavy	180	120	120	120
	Jumbo	180	180	120	120

표 A.5. 인접 활주로에서의 '선행 출발 – 후행 도착' 최소 분리 시간 (단위: 초)

		Trail			
		Light	Medium	Heavy	Jumbo
Lead	Light	80	52	45	45
	Medium	80	52	45	45
	Heavy	80	52	45	45
	Jumbo	80	52	45	45

부록 B. 수정 스케줄링 알고리즘

수정 스케줄링의 propagation은 일정 주기로 분할 수행된다. 부록 B는 5.2.에서 다루었던 한 주기에서의 상태 업데이트가 완료된 이후 propagation 과정을 정리하였다.

그림 5.14와 같이 상태 업데이트가 완료되면, 스케줄러는 다시 항공기가 위치한 가상 노드 V를 출발 노드로 삼아 propagation을 수행한다. 그림 B.1 ~ B.7은 스케줄러의 propagation 과정을 순서대로 나열한 것으로, 링크 아키텍처를 제외한 전체적인 과정은 2.4.와 동일하다.

그림 B.8 및 그림 B.9는 항공기가 이동 중 잠시 멈추는 '홀딩(Holding)' 상황을 보여준다. 재스케줄링 이후 항공기 현재 위치에서 추가 지연이 발생하였을 경우, 항공기는 지연된 시간동안 제자리에 멈춰 있다가 움직인다. 그러나 활주로를 이동하는 항공기는 활주로를 벗어날 때까지 추가 지연을 무시하고 이동한다.

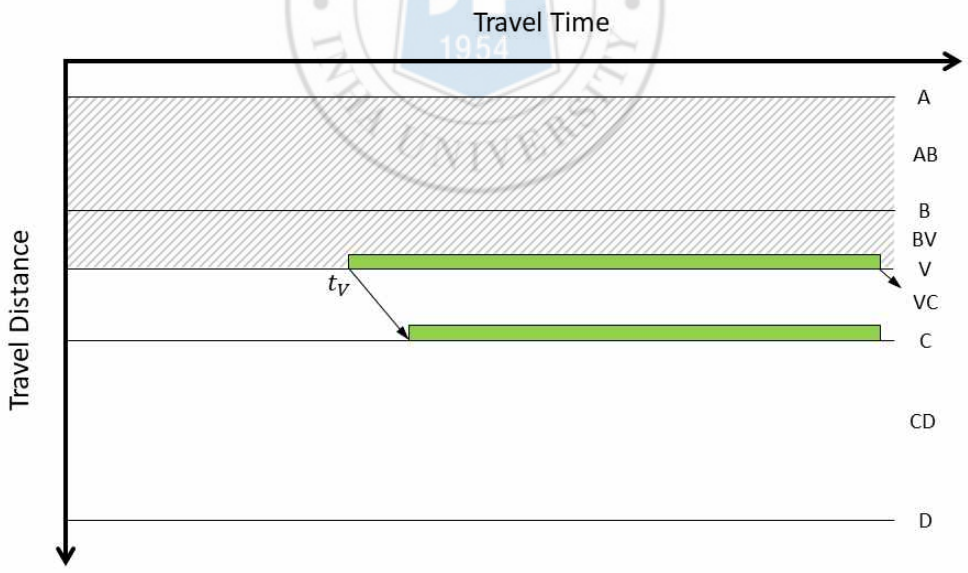


그림 B.1. 항공기의 현재 위치로부터 노드 C까지 스케줄을 전개한 모습

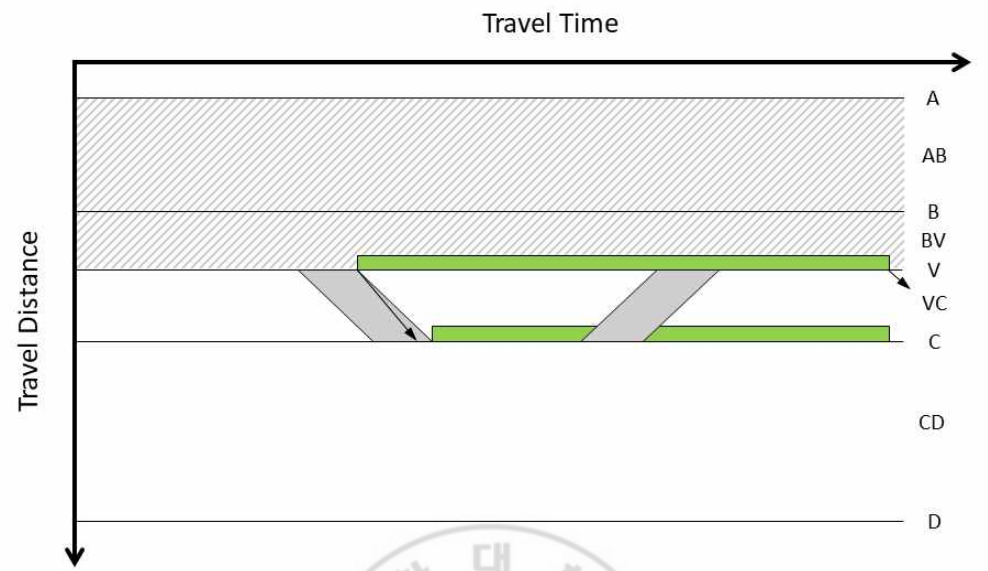


그림 B.2. 링크 VC의 제약 조건이 적용된 모습

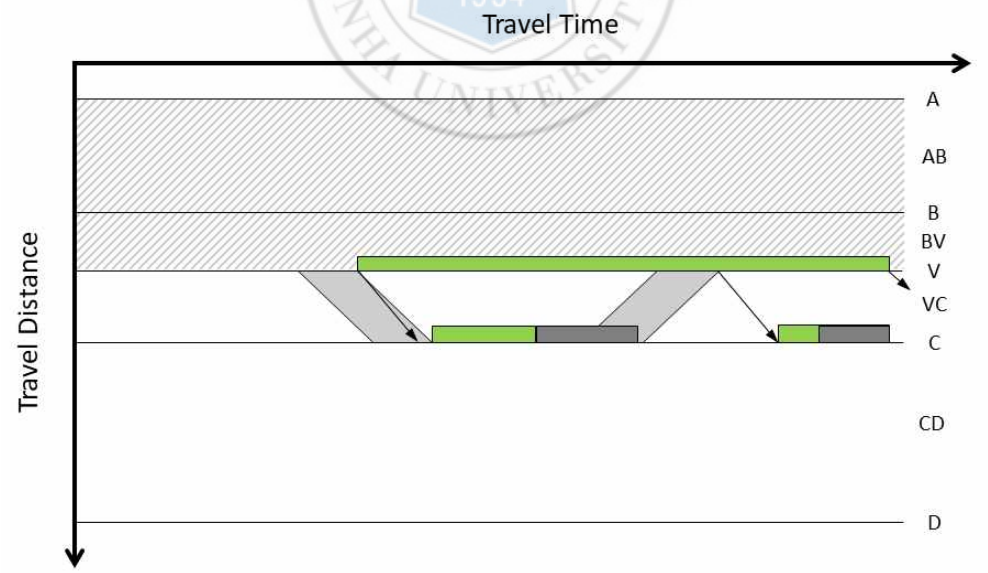


그림 B.3. 노드 C의 available entry slots

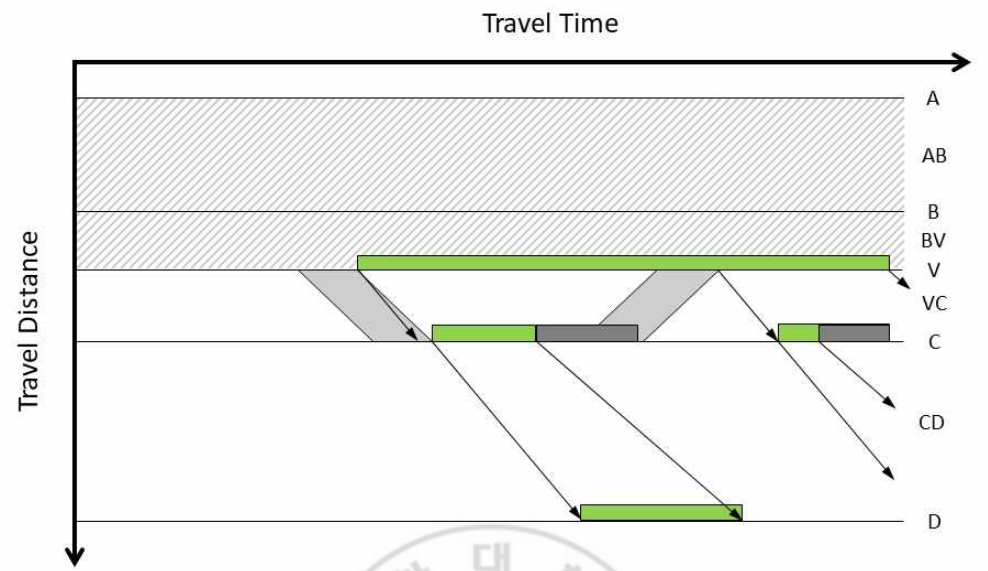


그림 B.4. 노드 C에서 도착 노드 D까지 스케줄을 전개한 모습

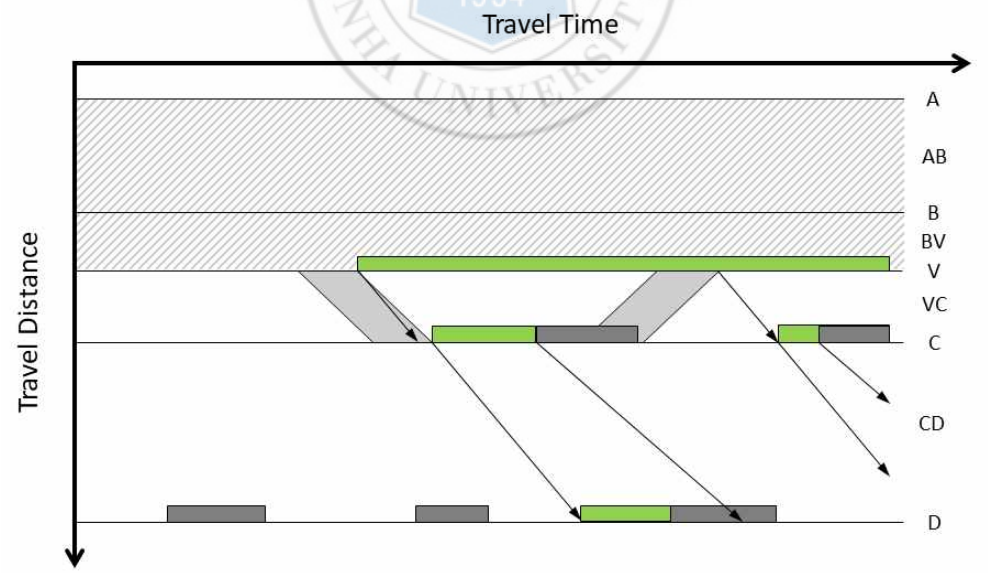


그림 B.5. 도착 노드 D의 available entry slots

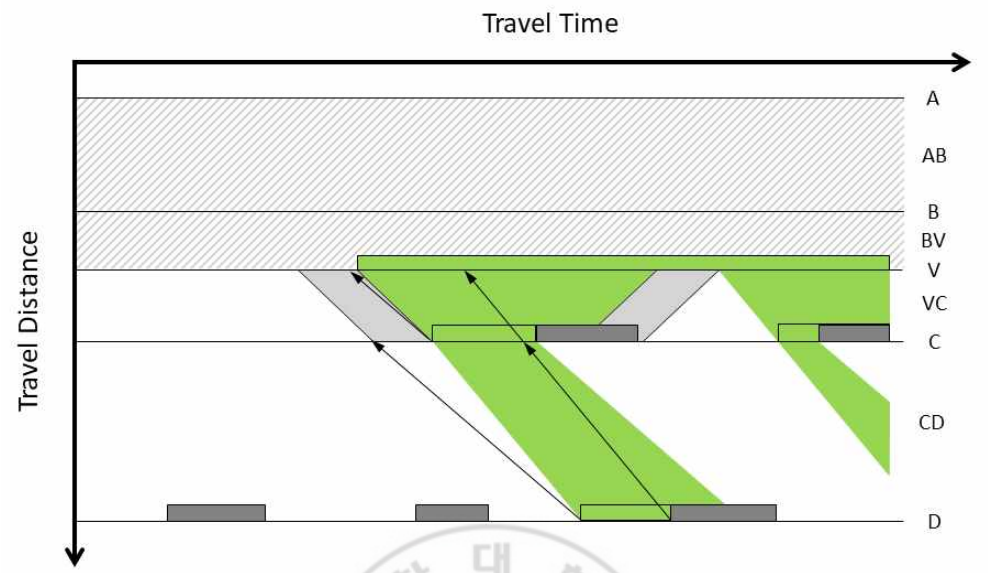


그림 B.6. 도착 노드 D부터 출발 노드 V까지의 backward propagation

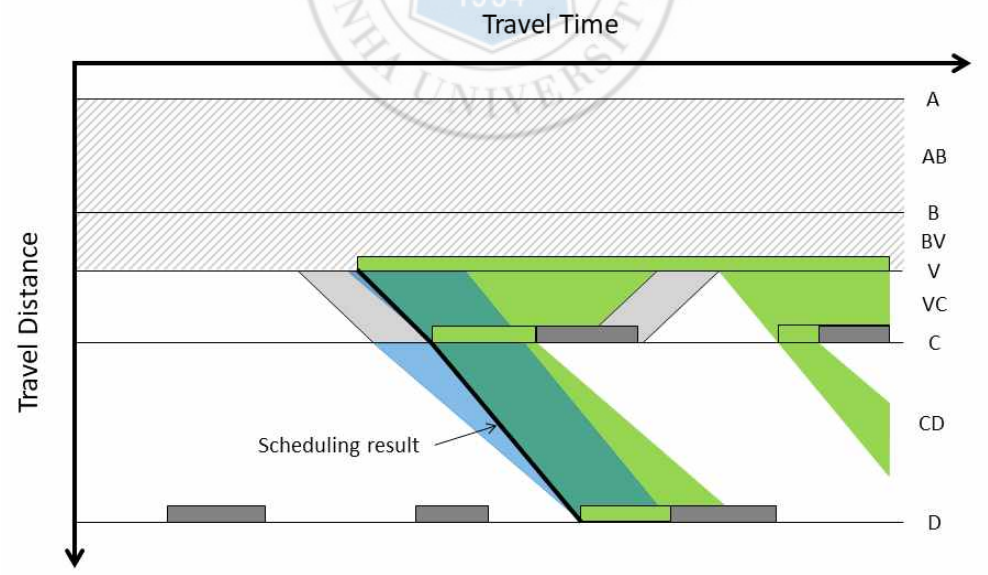


그림 B.7. Propagation 완료 후 스케줄링 결과

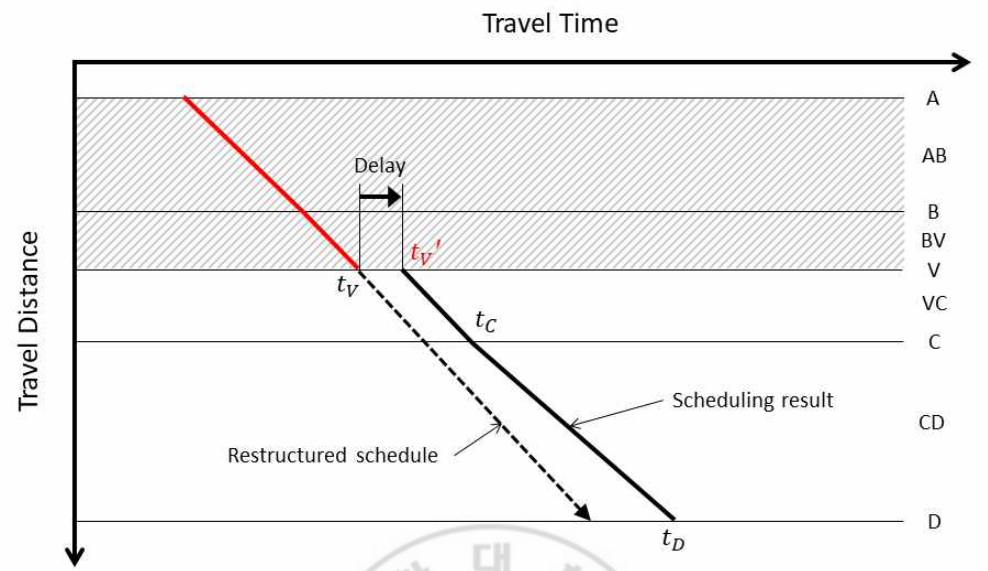


그림 B.8. Propagation 이후 현재 위치에서 추가 지연이 발생한 경우

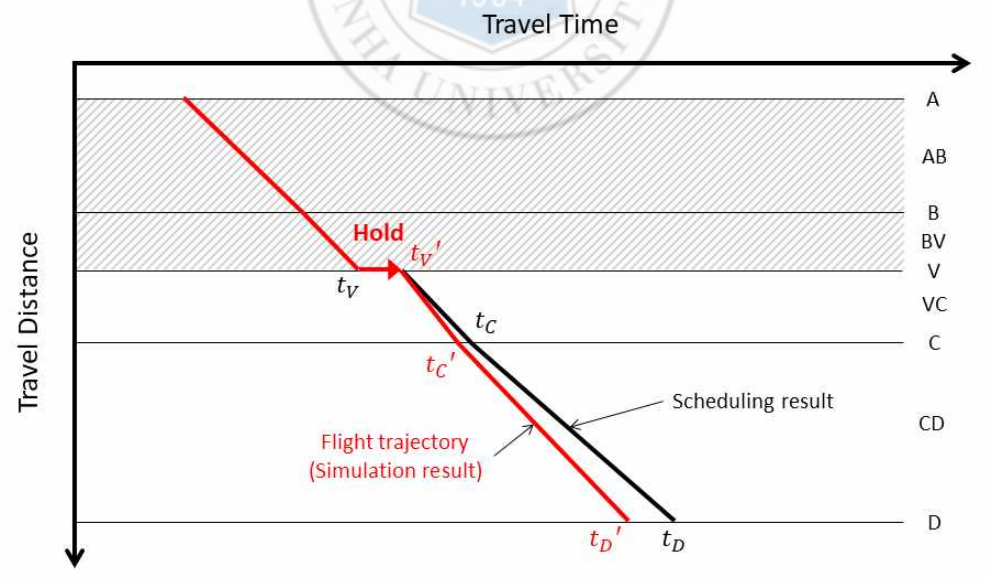


그림 B.9. 추가 지연으로 인한 holding

참고 문헌

[1] 한국항공협회, “항공통계(세계편),” 2018.

[2] FAA, “Trends in Accidents and Fatalities in Large Transport Aircraft, DOT/FAA/AR-10/16,” National Technical Information Services, 2010.

[3] ICAO, “ICAO Doc-4444, Procedures for Air Navigation Services—Air Traffic Management (PANS-ATM) 16th Edition,” 2016.

[4] De Neufville, R., Odoni, A., “Airport Systems : Planning, Design, and Management 2nd Edition,” McGraw-Hill, 2013.

[5] 한재현, 이금진, 안미진, 김연명, “연구총서 2010-26, 차세대 항공교통관리시스템(NGAS) 도입을 위한 기초연구,” 한국교통연구원, 2010.

[6] ICAO, “2016-2030 Global Air Navigation Plan, Doc 9750-AN/963 5th Edition,” 2016.

[7] JPDO, “Concept of Operations for the Next Generation Air Transportation System Version 3.2,” 2011.

[8] SESAR Consortium, “SESAR Master Plan D5,” 2008.

[9] 정명숙, 은연주, 오은미, 전대근, “항공기 출발 관리 기술 및 시스템 개발 동향,” 항공우주산업기술동향, Vol. 13, No. 2, pp.220-229, 2015.

[10] 국토교통부, “Aviation System Block Upgrades 국내적용을 위한 기본 기획 연구 – 항공기 출발 및 도착 통합관리 기술 연구,” 국가과학기술정보센터, 2014.

[11] Zelinski, S., “A Framework for Integrating Arrival, Departure, and Surface Operations Scheduling,” 2014 IEEE/AIAA 33rd Digital Avionics Systems Conference (DASC), 2014.

[12] EUROCONTROL, “The Manual: Airport CDM Implementation Version 5.0,” 2017.

[13] SESAR Joint Undertaking, “Basic DMAN Operational Service and Environment Definition (OSED),” 2011.

[14] Jung, Y., “Overview - Spot and Runway Departure Advisor(SARDA),” NASA Ames Research Center, August 2014.

[15] Jung, Y., Hoang, T., Montoya, J., Gupta, G., Malik, W., and Tobias, L., “A Concept and Implementation of Optimized Operations of Airport Surface Traffic,” 10th AIAA Aivation Technology, Integration, and Operations (ATIO) Conference, AIAA, Fort Worth, TX, September 2010.

[16] Jung, Y., Hoang, T., Montoya, J., Gupta, G., Malik, W., Tobias, L., and Wang, H., “Performance Evaluation of a Surface Traffic Management Tool for Dallas/Fort Worth International Airport,” Ninth USA/Europe Air Traffic Management Research and Development Seminar (ATM2011), 2011.

[17] Hoang, T., Jung, Y., Holbrook, J. B., and Malik, W., “Tower Controllers’ Assessment of the Spot and Runway Departure Advisor(SARDA) Concept,” Ninth USA/Europe Air Traffic Management Research and Development Seminar (ATM2011), 2011.

[18] Hayashi, M., Hoang, T., Jung, Y., Gupta, G., Malik, W., and Dulchinos, V. L., “Usability Evaluation of the Spot and Runway Departure Advisor(SARDA) Concept in a Dallas/Fort Worth Airport Tower Simulation,” Tenth USA/Europe Air Traffic Management Research and Development Seminar (ATM2013), 2013.

[19] Malik, W., Lee, H., and Jung, Y., “Runway Scheduling for Charlotte Douglas International Airport,” 16th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference, AIAA, Washington, DC, 2014.

[20] Coupe, W.J., Lee, H., Jung, Y., Chen, L., and Robeson, I.J., "Scheduling Improvements Following the Phase 1 Field Evaluation of the ATD-2 Integrated Arrival, Departure, and Surface Concept," the 13th USA/Europe Air Traffic Management Research and Development Seminar (ATM2019), 2019.

[21] Republic of Korea, "Improved Airport Operation through A-CDM and AMAN/DMAN Implementation of the Republic of Korea," 55th Conference of Directors General of Civil Aviation Asia and Pacific Regions, Nadi, Fiji, October 2018.

[22] Eun, Y., Jeon, D., Lee, H., Zhu, Z., Jung, Y., Jeong, M., Kim, H., Oh, E., Hong, S., and Lee, J., "Operational Characteristics Identification and Simulation Model Validation for Incheon International Airport," 16th AIAA Aviation Technology, Integration and Operations (ATIO) Conference, AIAA, Washington, DC, 2016.

[23] Eun, Y., Jeon, D., Lee, H., Jung, Y., Zhu, Z., Jeong, M., Kim, H., Oh, E., and Hong, S., "Optimization of Airport Surface Traffic: A Case-study of Incheon International Airport," 17th AIAA Aviation Technology, Integration and Operations (ATIO) Conference, AIAA, Denver, CO, 2017.

[24] Balakrishnan, H., and Chandran, B., "Scheduling Aircraft Landings under Constrained Position Shifting," AIAA Guidance, Navigation, and Control Conference and Exhibit, AIAA, Keystone, CO, 2006.

[25] Lindsay, K.S., Boyd, E.A., and Burlingame, R., "Traffic Flow Management Modeling with the Time Assignment Model," Air Traffic Control Quarterly Vol. 1, No. 3, 1993, pp.255-276.

[26] Bertsimas, D., and Stock-Patterson, S., "The Air Traffic Flow Management with Enroute Capacities," Operations Research, Vol. 46, No. 3, 1998, pp. 406-422.

[27] Hong, Y., Choi, B., Lee, K., and Kim, Y., "Dynamic Robust Sequencing and Scheduling Under Uncertainty for the Point Merge System in Terminal

Airspace," IEEE Transactions on Intelligent Transportation Systems, Vol. 19, No. 9, 2017, pp. 2933-2943.

[28] Hong, Y., Lee, S., Lee, K., and Kim, Y., "Optimal Scheduling Algorithm for Air Traffic Point Merge System Using MILP," Advances in Aerospace Guidance, Navigation and Control, Springer, Cham, 2018, pp. 407-420.

[29] Lee, S., Hong, Y., and Kim, Y., "Optimal Scheduling Algorithm in Point Merge System Including Holding Pattern based on Mixed-Integer Linear Programming," Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering, 2019, 0954410019830172.

[30] Visser, H.G., and Roling, P.C., "Optimal airport surface traffic planning using Mixed Integer Linear Programming," 3rd AIAA Aviation Technology, Integration, and Operations (ATIO) Conference, AIAA, Denver, CO, November 2003.

[31] Smeltink, J.W., M.J. Soomer, De Waal, P.R., Van Der Mei, R.D., "An Optimisation Model for Airport Taxi Scheduling," 2004.

[32] Rathinam, S., Montoya, J., Jung, Y., "An Optimization Model for Reducing Aircraft Taxi Times at the Dallas-Fort Worth International Airport," 26th International Congress of the Aeronautical Sciences (ICAS), 2008.

[33] Montoya, J., Wood, Z., Rathinam S., and Malik, W., "A Mixed Integer Linear Program for Solving a Multiple Route Taxi Scheduling Problem," AIAA Guidance, Navigation, and Control Conference, Toronto, ON, August 2010.

[34] Gupta, G., Malik, W., and Jung, Y.C., "A Mixed Integer Linear Program for Airport Departure Scheduling," the 9th AIAA Aviation Technology, Integration, and Operation (ATIO) Conference, AIAA, Hilton Head, SC, September 2009.

[35] Malik, W.A., Lee, H., and Jung, Y.C., "Runway Scheduling for Charlotte Douglas international Airport," the 16th AIAA Aviation Technology,

Integration, and Operation (ATIO) Conference, AIAA, Washington, DC, June 2016.

[36] Clare, G.L., and Richards, A.G., "Optimization of Taxiway Routing and Runway Scheduling," IEEE Transactions on Intelligent Transportation Systems Vol. 12, No. 4, 2011, pp. 1000-1013.

[37] Lee, H., and Balakrishnan, H., "A Comparison of Two Optimization Approaches for Airport Taxiway and Runway Scheduling," the 31st IEEE/AIAA 31st Digital Avionics Systems Conference (DASC), IEEE, 2012.

[38] Malik, W.A., and Jung, Y.C., "Exact and Heuristic Algorithms for Runway Scheduling," the 16th AIAA Aviation Technology, Integration, and Operation (ATIO) Conference, AIAA, Washington, DC, June 2016.

[39] Eun, Y., Jeon, D., Kim, H., Jung, Y., Lee, H., Zhu, Z., and Hosagrahara, V., "A Tactical Scheduler for Surface Metering under Minimum Departure Interval Restrictions," the 38th IEEE/AIAA Digital Avionics Systems Conference (DASC), IEEE, San Diego, CA, September 2019.

[40] Meyn, L.A., "A Closed-Form Solution to Multi-Point Scheduling Problem," AIAA Modeling and Simulation Technologies (MST) Conference, AIAA, Toronto, Canada, August 2010.

[41] Palopo, K., Chatterji, G.B., and Lee, H., "Interaction of Airspace Partitions and Traffic Flow Management Delay," 10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference, AIAA, Fort Worth, TX, September 2010.

[42] Lee, H., Chatterji, G.B., and Palopo, K., "Interaction of Airspace Partitions and Traffic Flow Management Delay with Weather," 30th Digital Avionics Systems Conference (DASC), IEEE, Seattle, DC, October 2011.

[43] Park, C., Lee, H., and Meyn, L.A., "Computing Flight Departure Times using an Advanced First-Come First-Served Scheduler," 12th AIAA Aviation

Technology, Integration, and Operations (ATIO) Conference, AIAA, Indianapolis, ID, September 2012.

[44] Weisstein, E.W., "Interval," from MathWorld--A Wolfram Web Resource, https://mathworld.wolfram.com/Interval.html, Accessed on April 24, 2020.

[45] Meyn, L.A., and Erzberger, H., "Airport Arrival Capacity Benefits Due to Improved Scheduling Accuracy," 5th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference, AIAA, Arlington, VA, September 2005.

[46] Bender, E.A., and Williamson, S.G., "Lists, Decisions and Graphs : With an Introduction to Probability," 2010.

[47] Park, B., and Lee, H., "Development of a First-Come First-Served Departure Scheduler," 7th Asia-Pacific International Symposium on Aerospace Technology (APISAT), Cairns, Australia, 2015.

[48] Park, B., Lee, H.(Hyeonwoong), Lee, Kang, S., and Lee, H.(Hak-Tae), "Airport Surface Movement Scheduling with Route Assignment using First-Come First-Served Approach," 17th AIAA Aviation Technology, Integration, and Operation (ATIO) Conference, AIAA, Denver, CO, USA, June 2017.

[49] Park, B., Lee, H.(Hyeonwoong), and Lee, H.(Hak-Tae), "Extended First-Come First-Served Scheduler for Airport Surface Operation," International Journal of Aeronautical and Space Sciences (IJASS), Vol. 19, No. 2, 2018, pp. 509-517.

[50] FAA, Air Traffic Control, FAA Order JO 7110.65W, Federal Aviation Administration, 10 Dec. 2015.

[51] Dijkstra, E.W., "A Note on Two Problems in Connexion with Graphs," Numerische Mathematik, Vol. 1, No. 1, 1959, pp. 269-271.

[52] Yen, J.Y., "Finding the K Shortest Loopless Paths in a Network," Management Science, Vol. 17, No. 11, 1971, pp. 712-716.

[53] Park, B., Lee, H.(Hyeonwoong)., Lee, H.(Hak-Tae)., Eun, Y., Jeon, D., Zhu, Z., Lee, H.(Hanbong)., and Jung., Y.C., "Comparison of First-Come First-Served and Optimization Based Scheduling Algorithms for Integrated Departure and Arrival Management," 18th AIAA Aviation Technology, Integration, and Operation (ATIO) Conference, AIAA, Atlanta, GA, June 2018.

[54] 김태영, 박배선, 이현웅, 이학태, “항공기 지상 이동 Fast-Time 시뮬레이터 개발,” 한국항행학회논문지, Vol. 23. No. 1, 2019, pp. 1-7.