Journal of Korean Society of Industrial and Systems Engineering
https://doi.org/10.11627/jksie.2024.47.2.048

Vol. 47, No. 2 : 48-56, June 2024
ISSN : 2005-0461 / EISSN : 2287-7975

Branch and Bound Algorithm for Single Machine Scheduling with Step-Improving Jobs

Jun-Ho Lee†

School of Business, Chungnam National University

계단형 향상 작업을 갖는 단일설비 스케줄링을 위한 분기한정 알고리즘

이준호†

충남대학교 경상대학 경영학부

We examine a single machine scheduling problem with step-improving jobs in which job processing times decrease step-wisely over time according to their starting times. The objective is to minimize total completion time which is defined as the sum of completion times of jobs. The total completion time is frequently considered as an objective because it is highly related to the total time spent by jobs in the system as well as work-in-progress. Many applications of this problem can be observed in the real world such as data gathering networks, system upgrades or technological shock, and production lines operated with part-time workers in each shift. Our goal is to develop a scheduling algorithm that can provide an optimal solution. For this, we present an efficient branch and bound algorithm with an assignment-based node design and tight lower bounds that can prune branch and bound nodes at early stages and accordingly reduce the computation time. In numerical experiments well designed to consider various scenarios, it is shown that the proposed algorithm outperforms the existing method and can solve practical problems within reasonable computation time.

Keywords : Scheduling, Single Machine, Step-Improving Jobs, Branch and Bound Algorithm

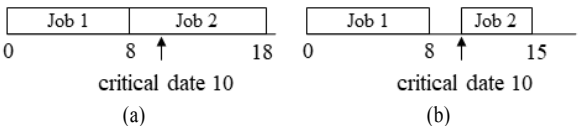
1. 서 론

본 논문에서는 계단형 향상(step-improving) 작업을 갖는 단일설비(single machine) 스케줄링 문제를 다룬다. 계단형 향상 작업이란 중요시점들(critical dates)을 기준으로 작업 시간이 계단형으로 감소하는 특성을 갖는 작업들을 일컫는다. 이러한 특성은 다양한 산업문제에서 찾아볼 수 있다. 통신 네트워크에서 데이터를 전송하는 경우 압축을 하지 않고 전송하면 전송시간이 길지만 일정 시간이 소요되는 압축을 수행한 후 데이터를 전송하면 전송시간이 감소하게 된다. 이 때 데이터 전송을 담당하는 서버를 단일설비로, 데이터 전송시간을 작업시간으로, 압축이 완료되는 시점을 중요시점으로 볼 수 있다[1, 10]. 또한, 압축률에 따라 압축에 필요한 시간이 상이할 수 있으므로 복수의 중요시점들이 존재할 수 있다. 다른 예로는 교대(shift)를 사용하는 제조 산업을 생각해 볼 수 있다. 3교대로 생산이 이루어지는 경우 학습효과, 추가인원 투입 등을 통해 교대가 진행될수록 작업시간이 감소하는 경우를 쉽게 관찰할 수 있다[6]. 이 외에도 주기적인 설비의 하드웨어 및 소프트웨어 업그레이드, 대규모 정비(overhaul), 기술적 진보 등을 통해 특정 시점이 지난 후에 작업시간이 감소하는

Received 2 April 2024; Finally Revised 17 April 2024;
Accepted 18 April 2024
† Corresponding Author : junholee@cnu.ac.kr

경우 또한 본 연구에서 다루는 문제의 대표적인 예라고 할 수 있다[7, 8].

목적식으로는 총완료시간(total completion time) 최소화를 고려한다. 총완료시간이란 모든 작업의 작업종료 시각의 합을 의미하며, 이는 결국 작업물들이 시스템에 머무르는 시간의 총합을 의미하므로 재공재고(work-in-progress, WIP)와도 관련이 깊다. 이러한 이유로 총완료시간 최소화는 생산 스케줄링 문제에서 자주 다루어지는 목적식 중 하나이다[4, 6, 7, 8]. 따라서 본 논문에서는 단일설비에서 총완료시간을 최소화하기 위해 복수의 작업들을 어떠한 순서로 어떠한 시점에 생산할지 결정하는 문제를 다루며, 이를 해결하기 위해 분기한정(branch and bound; B&B) 알고리즘을 개발하여 최적해(optimal solution)를 도출하는 것을 목표로 한다. 고정된 작업시간을 가정한 대부분의 기존 단일설비 스케줄링 연구들은 주어진 목적식 하에서 작업의 순서만을 결정하는 문제를 다루었다. 하지만 계단형 향상 작업과 같이 작업 개시 시각에 따라 작업시간이 변동되는 경우에는 작업순서 뿐 아니라 각 작업의 시작 시각 또한 결정해야 한다. 이는 기존 단일설비 스케줄링 문제들과의 큰 차이점이라고 할 수 있다. 일례로 아래 <Figure 1>과 같이 작업시간이 각각 8과 10인 두 개의 작업이 중요시점 10을 기준으로 중요시점 또는 그 이후에 작업이 시작될 경우 작업시간이 반으로 줄어드는 경우를 고려해보자. 앞서 소개한 데이터 전송 예에 적용해보면 Job 1과 Job 2는 전송해야하는 데이터를 의미하고, 중요시점 10은 데이터를 압축했을 때 압축이 완료되는 시점을 의미한다. 즉, Job 2를 압축 없이 전송한다면 10이라는 시간이 전송에 소요되지만, 압축 후 전송할 경우 전송시간이 절반인 5가 된다.



<Figure 1> An Example of Step-improving Jobs

<Figure 1>의 (a)의 경우 작업시간 변동을 고려하지 않을 때 얻을 수 있는 일반적인 스케줄, 즉, 작업 간 유휴시간 없이 작업시간이 작은 순서대로 공정을 진행하는 경우를 의미한다. 이때 총완료시간은 26이 된다. <Figure 1>의 (b)의 경우 작업의 계단형 향상특성을 고려하여 의도적인 유휴시간을 갖고 두 번째 작업이 중요시점에 시작하도록 한 스케줄이며, 이 때 얻을 수 있는 총완료시간은 23으로 (a)의 스케줄보다 더 작은 총완료시간을 얻을 수 있다. 즉, 전술한 바와 같이 본 연구에서 다루는 문제의 경우 작업순서 뿐 아니라 작업의 시작시점까지 결정해야하는 복잡한 문제라고 할 수 있다.

본 연구에서 다루는 문제와 같이 작업시작 시각에 따라 작업시간이 변경되는 특성을 갖는 문제를 시간종속적(time-dependent) 스케줄링 문제라고 하며, 지금까지 다수의 연구가 수행되었다. 시간종속적 스케줄링 범주에 속하는 특성으로는 본 연구에서 다루는 계단형 향상 작업을 포함하여 작업시작 시각에 따라 작업시간이 계단형으로 증가하는 경우(step- deteriorating), 작업시간이 선형으로 증가하거나 감소하는 경우 등 다양한 형태가 존재한다[2, 9, 11]. 이에 대한 최신 리뷰는 Gawiejnowicz[4]를 참고할 수 있다. 본 연구의 핵심적인 스케줄링 요구사항이라고 할 수 있는 계단형 향상 작업을 다룬 연구들을 살펴보면, Cheng et al.[3]은 하나의 중요시점을 고려하였으며, 최대완료시간(makespan) 최소화를 위한 알고리즘을 제시하였다. 본 연구에서는 복수의 중요시점을 고려하며 총완료시간 최소화를 목적식으로 한다는 점에서 차이가 있다. Ji et al.[5] 또한 Cheng et al.[3]과 유사하게 최대완료시간 최소화를 위한 효율적인 스케줄링 알고리즘을 제시하였다. Kim and Oron[7]은 하나의 중요시점을 가정하여 총완료시간 최소화를 위한 스케줄링 문제를 다루었다. 해당 문제의 계산복잡도가 매우 높음을, 즉, NP- hard임을 증명하였고 최적해 도출을 위한 혼합정수계획모형(mixed integer linear programming; MILP)을 제시하고 이를 바탕으로 휴리스틱 알고리즘 또한 개발하였다. 고려한 목적식은 본 연구와 동일하지만 단수의 중요시점을 가정하였다는 점에서 문제의 난이도와 복잡도가 본 연구에서 다루는 문제보다 낮다고 할 수 있다. Kim et al.[8]은 유사한 문제에 대해 총가중완료시간(total weighted completion time) 최소화를 목적식으로 고려하였다. 해당 문제를 해결하기 위해 동적계획법(dynamic programming) 기반의 알고리즘을 제시하였으며 추가로 효율적인 휴리스틱 알고리즘 또한 개발하였다. 본 연구에서 다루는 목적식과 상이한 목적식을 고려하였으며 단수의 중요시점을 고려했다는 점에서 차이가 있다. 최근 Kim et al.[6]은 최초로 복수의 중요시점들을 고려하였으며 총완료시간 최소화를 위한 스케줄링 문제를 다루었다. 해당 연구에서 다루는 문제의 계산 복잡도가 NP-hard임을 증명하였고, 최적해 도출을 위한 혼합정수계획모형 및 휴리스틱 알고리즘을 제시하였다. 본 연구에서 다루는 문제와 동일한 가정 하에서 최적해 도출을 위한 혼합정수계획모형을 제시하였지만 문제의 계산복잡도가 높아 작은 크기의 문제들만을 해결할 수 있다는 한계가 있다.

전술한 바와 같이 계단형 향상 작업을 다룬 연구들이 존재하지만 본 연구에서 다루는 문제와 스케줄링 요구사항이 다른 연구들이 대부분이며, 동일한 문제를 다룬 Kim et al.[6]의 경우 최적해를 도출할 수 있는 문제의 크기에 한계가 있다. 따라서 본 연구에서는 효율적인 분기한정 알

고리즘을 새롭게 제시하여 최적해 도출을 위한 계산시간을 줄이고 해결할 수 있는 문제의 범위를 확장하고자 한다.

제2장에서는 본 연구에서 다루는 문제를 자세하게 설명하고 문제 모델링 및 해결을 위한 혼합정수계획모형을 소개한다. 제3장에서는 본 연구의 핵심 결과라고 할 수 있는 분기한정 알고리즘을 소개하며, 제4장에서는 계산실험을 통해 개발된 분기한정 알고리즘의 성능을 검증한다. 마지막으로 제5장에서는 결론을 제시하며 향후 연구 방향에 관해 논한다.

2. 문제정의 및 혼합정수계획모형

본 연구에서는 n개의 작업, 즉, $J=\{1,2,...,n\}$을 가정하며 각각의 작업은 기본 작업시간 p_j, $j\in\{1,2,...,n\}$,를 갖는다. Non-preemptive 작업을 가정하며 이는 단일설비가 특정 작업을 한번 시작하면 완료 전까지 중간에 멈추고 다른 작업을 수행할 수 없음을 의미한다. 각 작업의 실제 작업시간은 m개의 중요시점 $D_1, D_2, ..., D_m$과 작업시작 시점에 의해 결정되며 계단형으로 감소한다. 이를 수식으로 표현하면 특정 작업 j가 $D_{i-1} \le t < D_i$, $i\in\{1,2,...,m+1\}$,를 만족하는 시점 t에 시작될 경우 실제 작업시간은 $\delta_i \times p_j$가 된다. 이 때 전술한 바와 같이 본 연구에서는 계단형 향상 작업을 가정하므로 $\delta_1 > \delta_2 > \cdots > \delta_{m+1}$을 만족하며, D_0과 D_{m+1}은 각각 0과 ∞을 의미한다. 또한, p_j가 작업 j의 기본 작업 시간임을 반영하기 위해 $\delta_1 = 1$로 설정한다. 표현의 편의를 위해 구간(period)이라는 용어를 사용하며, 구간 i는 $[D_{i-1}, D_i)$을 의미한다.

특정 스케줄 σ는 단일설비에서 이루어지는 작업의 순서와 작업별 시작 시점 정보를 포함한다. 일반적인 단일설비 스케줄링 문제에서는 스케줄이 작업순서만을 나타내는 경우가 대부분이나, 본 연구에서 다루는 문제는 <Figure 1>의 (b)의 예와 같이 작업 간 의도적인 유휴시간을 통해 더 좋은 해를 얻을 수 있는 경우가 있기에 작업 시작 시점 정보 또한 포함해야 한다. $S_j(\sigma)$와 $C_j(\sigma)$는 스케줄 σ 하에서 작업 j의 시작 시각과 종료 시각을 각각 나타내며, 스케줄을 특정할 필요가 없는 경우에는 간단히 S_j와 C_j로 표현한다. 본 논문에서는 목적식으로 총완료시간 최소화를 고려하므로 $\sum_{j=1}^{n} C_j$을 최소로 하는 최적 스케줄 σ^*를 도출할 수 있는 분기한정 알고리즘의 개발을 목표로 한다.

제안하는 분기한정 알고리즘은 다음 장에서 자세히 소개할 예정이며, 그 전에 본 논문에서 다루는 문제에 대한 혼합정수계획모형을 소개한다. 1장에서 소개한 바와 같이 본 연구에서 다루는 문제에 대한 혼합정수계획모형은 Kim et al.[6]에 의해 개발되었다. 본 논문을 제외하면 Kim et al.[6]의 혼합정수계획모형이 최적해를 도출할 수 있는 유일한 방법이므로, 향후 4 장에서 본 연구를 통해 개발된 분기한정 알고리즘과 Kim et al.[6]의 혼합정수계획모형의 성능을 비교할 예정이다. 비록 본 연구에서 새롭게 제시하는 모형은 아니지만 전술한 바와 같이 본 연구와 관련이 깊고 본 논문의 자체완결성을 높이기 위해 Kim et al.[6]의 혼합정수계획모형을 소개한다.

일반성을 잃지 않고, 작업들이 $p_1 \le p_2 \le \cdots \le p_n$을 만족하도록 정렬되어 있다고 가정하면, 아래의 혼합정수계획모형과 Gurobi 또는 Cplex와 같은 상용 소프트웨어를 통해 최적해를 도출할 수 있다.

결정변수(decision variable)

- S_j: 작업 j의 시작 시각
- C_j: 작업 j의 종료 시각
- x_{ij}: 작업 j의 시작 시각이 $[D_{i-1}, D_i)$에 속하면 1, 그렇지 않으면 0의 값을 갖는 이진(binary) 변수

혼합정수계획모형(MILP)

$$\text{Minimize} \quad \sum_{j=1}^{n} C_j \tag{1}$$

subject to

$$\sum_i x_{ij} = 1, \qquad \forall j\in\{1,2,...,n\} \tag{2}$$

$$S_j + M(1-x_{ij}) \ge D_{i-1}, \quad \forall i\in\{1,2,...,m+1\},\ \forall j\in\{1,2,...,n\} \tag{3}$$

$$S_j + M(x_{ij}-1) < D_i, \quad \forall i\in\{1,2,...,m+1\},\ \forall j\in\{1,2,...,n\} \tag{4}$$

$$S_j - (S_k + \delta_i p_k) \ge M(x_{ij} + x_{ik} - 2), \quad \forall i\in\{1,2,...,m+1\}, \quad \forall j,k\in\{1,2,...,n\}, j>k \tag{5}$$

$$S_j - (S_k + \delta_i p_k) \ge M\left(\sum_{p=i+1}^{m+1} x_{pj} + x_{ik} - 2\right), \quad \forall i\in\{1,2,...,m+1\}, \quad \forall j,k\in\{1,2,...,n\} \tag{6}$$

$$C_j \ge S_j + \sum_i \delta_i p_j x_{ij}, \qquad \forall j\in\{1,2,...,n\} \tag{7}$$

$$x_{ij}\in\{0,1\}, S_j \ge 0, C_j \ge 0 \tag{8}$$

식 (1)은 본 연구에서 다루는 목적식인 총완료시간 최소화를 나타낸다. 식 (2)는 특정 작업이 하나의 구간에서만 시작할 수 있다는 제약을 나타낸다. 식 (3)과 (4)는 x_{ij}가 1인 경우 작업 j의 시작 시각이 구간 i에 속하도록 강제하는 역할을 한다. 이 때 M은 충분히 큰 양수를 의미한다. 식 (5)는 특정 구간에 할당된 작업들이 구간 내에서 기본 작업시간의 오름차순으로 공정이 이루어져야함을 나타낸

다. 시간 변동이 없는 경우 작업시간의 오름차순으로 공정을 진행하면 총완료시간 최소화를 달성할 수 있음이 알려져 있기에 이러한 속성을 적용한 것이다. 식 (6)은 $p > i$인 두 구간 p와 i에서 구간 p에 할당된 작업은 구간 i에 할당된 작업들이 모두 종료된 후에 시작할 수 있다는 제약을 나타낸다. 마지막으로 식 (7)은 작업 시작 시각과 종료 시각을 연결해주는 역할을 한다. 혼합정수계획모형의 더욱 자세한 설명은 Kim et al.[6]을 참고할 수 있다.

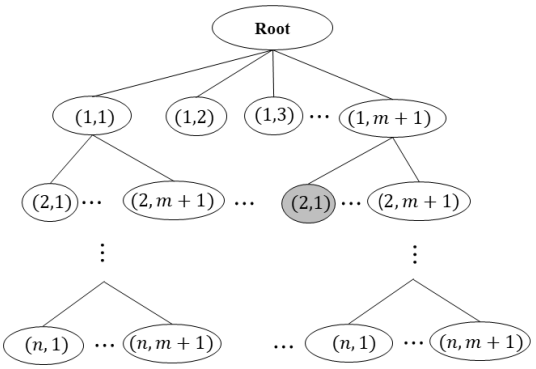
3. 분기한정 알고리즘

본 장에서는 앞서 2장에서 소개한 문제의 최적스케줄 도출을 위한 분기한정 알고리즘을 제시한다. 분기한정 알고리즘은 부분 또는 전체 스케줄(partial schedule)을 나타내는 노드(node)들을 탐색하며 최적해를 찾아가는 과정을 의미한다. 분기한정 알고리즘에서 중요한 개념으로는 노드, 목적식 값의 상한(upper bound; UB), 각 노드에서의 목적식 값 하한(lower bound; LB) 등이 있다.

말단 노드를 제외한 노드들은 일부의 작업들에 대해서만 스케줄이 결정된 부분 스케줄을 나타내도록 정의되어야 한다. 각 노드에서 현재까지 결정된 부분 스케줄을 바탕으로 나머지 작업들을 어떤 방식으로 스케줄링 하더라도 목적식의 값이 최소 *LB* 이상이 된다는 것을 보이면, *LB*는 해당 노드의 목적식 값 하한이 된다. 반면 스케줄링 문제의 목적식 값 상한을 *UB*라고 하면, 초기에 *UB*는 무한대의 값으로 둘 수도 있고 간단한 휴리스틱 알고리즘을 통해 스케줄을 얻고 해당 스케줄의 목적식 값을 계산하여 설정할 수도 있다. *UB*는 노드들을 탐색하며 완성된 스케줄을 얻을 때 마다 기존 값과 새로운 값을 비교하여 작은 값으로 업데이트 된다. 즉, 최적 스케줄의 목적식 값은 적어도 *UB* 이하라고 할 수 있다. 따라서 특정 노드에서 *LB*의 값이 *UB*보다 크다면 해당 노드의 부분 스케줄은 최적 스케줄로 이어질 수 없기에, 해당 노드의 후속(혹은 자식) 노드들은 더 이상 탐색하지 않는다. 이러한 과정을 가지치기(pruning)라 한다. 이러한 방식으로 탐색영역(search space)을 줄여가며 효율적으로 최적 스케줄을 탐색하게 된다. 따라서 분기한정 알고리즘의 성능을 높이기 위해 각 노드들을 어떻게 정의할 것인지, 각 노드에서 *LB*를 어떻게 도출할 것인지, 초기 *UB* 값을 어떻게 설정할 것인지가 중요한 요소들이 된다.

3.1 노드 정의

전술한 바와 같이 각 노드는(말단 노드 제외) 부분 스케줄을 나타내도록 정의되어야 하며, 노드의 정의에 따라 알고리즘 성능이 달라질 수 있다. 본 논문에서는 작업시간이 고정된 단일설비 스케줄링에서 최단작업시간우선(shortest processing time; SPT) 규칙을 통해 총완료시간을 최소화할 수 있다는 점과 특정 구간에 할당된 작업들의 경우 작업시간이 고정되어 있다고 볼 수 있는 점을 고려하여 각 노드들이 특정 작업이 어떤 구간에 할당되어 있는지를 나타내도록 정의하였다. 전술한 바와 같이 구간에 작업들이 할당되고 나면 구간 내에서 최단작업시간우선 규칙을 통해 작업 순서를 결정할 수 있다. 이러한 속성을 통해 <Figure 2>와 같이 간단한 형태로 노드를 정의할 수 있다.



<Figure 2> Node Design

앞서 혼합정수계획모형에서와 같이 일반성을 잃지 않고, 작업들이 $p_1 \le p_2 \le \cdots \le p_n$을 만족하도록 정렬되어 있다고 가정하자. <Figure 2>의 각 노드들은 2개의 숫자로 표시되며, 앞의 숫자는 작업을 뒤의 숫자는 해당 작업이 할당된 구간을 의미한다. 또한 추후에 역추적(backtracking)을 통해 전체 스케줄을 얻을 수 있도록 각 노드들은 부모 노드의 정보를 저장한다. 예를 들어 노드 (1,1)은 첫 번째 작업이 첫 번째 구간에 할당된 부분 스케줄을 의미한다. 다른 예로 회색으로 표시된 노드 (2,1)의 경우 부모노드가 $(1,m+1)$이므로 첫 번째 작업이 $m+1$ 구간에 그리고 두 번째 작업이 첫 번째 구간에 할당된 부분 스케줄을 의미한다. 이와 유사하게 말단 노드 (n,i), $i \in \{1,2,...,m+1\}$, 까지 탐색이 이루어진 경우 부모 노드들을 추적하여 전체 스케줄을 얻을 수 있게 된다.

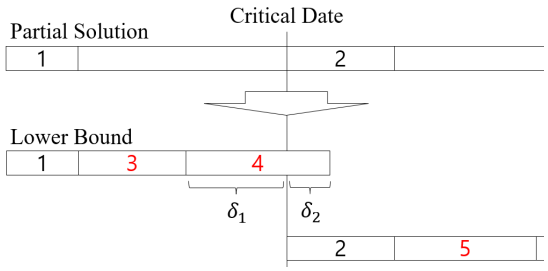
3.2 초기 목적식 값 상한(UB)

초기 목적식 값의 상한이 최적해의 목적식 값과 가까울수록 노드 탐색 중 가지치기의 가능성이 높아지며 따라서 탐색영역을 줄이고 빠른 시간 내에 최적해를 도출할 수 있다. 본 논문에서는 Kim et al.[6]에 의해 제안된 휴리스틱 알고리즘을 통하여 목적식의 값을 구하고 이를 초기 목적식 값 상한으로 사용한다. 해당 알고리즘은 최단작업시간

우선 규칙을 기본 아이디어로 사용하며, 중요시점 근처의 작업들에 대해 <Figure 1>의 예와 같이 의도적인 유휴시간을 가진 후 중요시점에 작업을 시작하는 경우와 유휴시간 없이 중요시점 전에 작업을 시작하는 경우를 비교하여 작업 종료시각이 더 작은 대안을 선택하는 형태로 스케줄을 도출한다.

3.3 노드의 목적식 값 하한(LB)

말단 노드를 제외한 노드들은 부분 스케줄을 의미하며 해당 부분 스케줄을 바탕으로 얻을 수 있는 스케줄의 목적식 값 하한 LB를 구하고, 목적식 값의 상한 UB와 비교하여 가지치기를 할지 계속 탐색할지를 결정하게 된다. 이때 LB가 느슨하면 효율적으로 가지치기를 할 수 없고 이는 알고리즘의 성능저하로 이어지므로 타이트한 LB를 개발하는 것이 매우 중요하다고 할 수 있다. 본 연구에서는 preemption과 구간 분리 아이디어를 통해 <Figure 3>과 같은 LB를 제안한다. 이해를 돕기 위해 한 개의 중요시점과 1부터 5의 다섯 개의 작업이 있는 간단한 예제를 고려하자. 또한 현재 노드는 (2,2)이며 부모 노드는 (1,1), 즉 부분 스케줄이 작업 1은 첫 번째 구간에 작업 2는 두 번째 구간에 할당된 경우를 가정하자.



<Figure 3> Lower Bound

<Figure 3>과 같이 아직 할당되지 않은 작업들을 순서대로 앞 구간에 할당한다. 즉, 이는 최단작업시간우선 규칙을 활용한다고 볼 수 있다. 작업 4와 같이 할당 후 구간을 넘어가는 경우 더 이상 해당 구간에 작업을 할당하지 않는다. 또한 실제 문제에서는 preemption이 허용되지 않지만 LB 계산 시에는 하한을 보장하기 위해 이를 허용한다. 즉, 작업 4의 경우 구간 1 내에서 작업이 이루어지는 경우에는 기본 작업시간에 δ_1을 곱하며, 넘어서는 작업에 대해서는 δ_2를 곱하여 작업시간을 산정하고 이를 통해 작업 종료시각을 계산한다. 따라서 <Figure 1>의 예와 같이 작업 간의 의도적인 유휴시간은 불필요하다. 또 다른 특징은 하한을 보장하기 위해 구간을 분리한다는 점이다. <Figure 3>의 작업 4와 같이 구간을 넘어서 작업이 종료되는 경우 실제 문제에서는 해당 작업이 끝난 후 다음 작업이 시작하지만 LB 계산 시에는 이를 무시한다. 즉 앞선 구간의 작업 종료와 관계없이 중요시점에 바로 첫 번째 작업이 시작됨을 가정한다. Preemptive 작업을 가정하면 최단작업시간우선 규칙으로 스케줄링 함으로써 총완료시간을 최소화할 수 있고 구간을 분리하여 특정 구간의 작업 시작 시각이 이전 구간에 영향을 받지 않도록 하였기 때문에 이러한 방식으로 계산된 총완료시간이 목적식 값의 하한이 됨은 자명하다.

노드 (a,b)에 대한 목적식 값 하한 계산은 다음의 의사코드(pseudo-code) 형태로 표현할 수 있다.

Step 1: 구간별 작업 종료 시점을 나타내는 F_i, $i=1,2,...,m+1$,을 정의하고 초기 값으로 $F_i = D_{i-1}$, $i=1,2,...,m+1$,로 설정한다. 각 작업의 종료시각을 나타내는 C_j, $j=1,2,...,n$, 를 정의한다.

Step 2: 노드 (a,b) 및 그 부모 노드를 포함하는 조상(ancestors) 노드를 추적하여 현재까지의 할당을 파악하고 순서대로 각 구간에 할당한다. 할당된 결과에 따라 F_i와 C_j를 업데이트 한다. 이때 preemption을 가정하며 구간을 넘어서는 작업에 대해서는 다음 구간의 δ값을 적용하여 F_i 및 C_j를 업데이트한다.

Step 3: 작업을 나타내는 인덱스 p와 구간을 나타내는 인덱스 q를 정의하고, 초기값으로 $p=a+1$, $q=1$을 할당한다.

Step 4: 만약 $p>n$이면 Step 6으로 간다. 그렇지 않으면 Step 5를 진행한다.

Step 5: 만약 $F_q \geq D_q$이면 q를 1만큼 증가시키고 Step 5를 다시 수행한다. 그렇지 않으면 작업 p를 구간 q에 할당하고 F_q와 C_p를 업데이트 한다. 이때 preemption을 가정하며 작업 p가 구간 q를 넘어서 종료될 경우 넘어서는 작업에 대해서는 δ_{q+1}값을 적용한다. p를 1만큼 증가시킨 후 Step 4로 간다.

Step 6: $\sum_j C_j$이 노드 (a,b)의 목적식 값 하한이 된다.

추가로 2장에서 소개한 혼합정수계획모형에서 부분 스케줄에서 결정된 할당에 대해 이진변수 x_{ij}의 값을 고정시키고, 아직 할당되지 않은 작업에 대해서는 x_{ij}가 0에서 1 사이의 실수값을 가질 수 있도록 relaxation하여 하한을 도출하는 방법도 개발하였다. 하지만 예비실험을 통해 앞서 소개한 하한에 비해 성능이 떨어짐을 확인하였기에 분기한정 알고리즘에 포함시키지 않았다.

3.4 알고리즘

본 연구에서 제안하는 최종 분기한정 알고리즘은 다음과 같다.

Step 1: 탐색할 노드들을 보관할 우선순위큐(priority queue) Q를 정의하자. Q 내에서 노드들은 노드의 목적식 값 하한(LB)의 오름차순으로 정렬된다. 초기 UB는 Kim et al.[6]의 휴리스틱 알고리즘을 통해 얻은 스케줄의 총완료시간으로 설정한다. 또한 노드 정보를 저장하기 위한 N^*를 정의하자.

Step 2: Q에 노드 $(1,1),(1,2),...,(1,m+1)$을 넣는다.

Step 3: Q가 비었으면 Step 7로 간다. 그렇지 않으면 Q의 맨 앞 노드를 꺼내고 이를 (a,b)라 하자. (a,b)를 역추적 하여 부분 스케줄을 도출한다. 해당 부분 스케줄이 feasible한지 확인한다. 즉, 해당 스케줄대로 할당할 경우 하나 이상의 작업이 할당된 구간에서 시작할 수 없다면 infeasible로 본다. 해당 부분 스케줄이 feasible일 경우 Step 4로 간다. 그렇지 않으면, Step 3을 다시 수행한다.

Step 4: 만약 $a=n$이면, Step 5로 간다. 그렇지 않으면 Step 6로 간다.

Step 5: 노드 (a,b)로부터 역추적을 통해 전체 스케줄을 도출하여 총완료시간을 구하고 이를 TCT라 하자. 만약 TCT가 UB보다 작으면, UB와 N^*를 각각 TCT와 (a,b)로 업데이트한다. Step 3으로 간다.

Step 6: 해당 노드 (a,b)의 목적식 값 하한 LB를 계산한다. 만약 LB가 UB보다 크면 Step 3으로 간다. 그렇지 않으면 해당 노드의 자식 노드들 즉, $(a+1,1)$, $(a+1,2)$,... , $(a+1,m+1)$을 생성하여 Q에 넣고 Step 3으로 이동한다.

Step 7: 알고리즘을 종료한다. 이때 최적 스케줄은 N^*를 역추적 함으로써 얻을 수 있고 해당 스케줄의 총완료시간이 최적 목적식의 값이 된다.

4. 계산실험

4.1 실험환경 및 디자인

본 연구에서 다루는 문제에 있어 문제의 난이도와 특성을 결정하는 중요 인자는 다음의 다섯 가지로 정리할 수 있다. 첫 번째는 작업 시간 p_j, $j\in\{1,2,...,n\}$, 이며 본 실험에서는 1에서 50 사이의 정수를 무작위로 생성하였다. 두 번째 인자는 작업수 n이며, 세 번째 인자는 중요시점의 수 m이다. 이때 n과 m이 커질수록 문제의 복잡도는 증가하게 되며, 특히 m이 증가함에 따라 문제의 복잡도는 기하급수적으로 증가하게 된다. 네 번째 인자는 구간 별 작업 시간 감소를 결정하는 δ_i, $i\in\{1,2,...,m+1\}$, 이다. 이에 대한 다양한 시나리오를 고려하기 위해 새로운 파라미터 $0<\alpha<1$를 사용하였으며, 구체적으로 주어진 α에 대해 $[\alpha,1]$ 내에서 $m+1$개의 난수를 생성하고 내림차순으로 정렬한 후 각각을 $\delta_1,\delta_2,...,\delta_{m+1}$ 값으로 사용하였다(단, $\delta_1=1$). 마지막으로 전체 작업시간 대비 구간의 길이 또한 중요한 인자라고 볼 수 있다. 극단적으로 각 구간의 길이가 매우 클 경우 한 구간 내에서 모든 작업을 수행할 수 있고 이는 계단형 향상 작업을 고려하지 않은 기존의 단일 설비 스케줄링 문제와 차이가 없다. 반면 각 구간의 길이가 너무 작을 경우 구간 내에서 오직 소수의 작업들만 수행할 수 있게 되며 문제의 복잡도가 높아질 가능성이 있다. 이러한 특성을 고려하기 위해 새로운 파라미터 β를 사용하였으며, 주어진 $0<\beta<1$ 하에서 각 중요시점 D_i, $i\in\{1,2,...,m\}$,를 $\beta\frac{i}{m}\left(\sum_{j=1}^{n}p_j\right)$로 설정하였다.

본 연구에서 개발된 분기한정 알고리즘은 Java 프로그래밍 언어를 통해 구현하였으며, 성능 비교를 위해 소개한 혼합정수계획모형(2장 참고)을 풀기위해 상용 소프트웨어인 Gurobi(version 11.0.1)를 사용하였다. 모든 실험은 PC(CPU: Intel Core i7-11800H, RAM: 32GB)에서 이루어졌다.

4.2 실험결과

앞서 언급한 바와 같이 기존 연구 결과 중 본 연구에서 다루는 문제에 대해 최적해를 제공하는 유일한 방법은 2장에서 소개한 Kim et al.[6]의 혼합정수계획모형이다. 따라서 본 연구에서 개발된 분기한정 알고리즘의 성능 실험을 위해 Kim et al.[6]의 혼합정수계획모형과의 비교실험을 수행하였다. Kim et al.[6]에 의해 증명되었듯이 본 연구에서 다루는 문제의 복잡도는 NP-hard(in the strong sense)이므로 큰 사이즈의 문제에 대해서는 $P\neq NP$인한 어떠한 방법으로도 최적해를 빠른 시간 내에 도출하는 것이 현실적으로 불가능하다. 따라서 본 실험에서는 적당한 크기의 문제를 가정하고 본 연구에서 제안한 방법과 기존 방법과의 계산시간 차이를 비교함으로써 제안된 방법의 성능을 검증하는 것을 목표로 한다. 구체적으로 $n\in\{10,20\}$, $m\in\{2,3\}$인 경우를 고려하였으며 앞서 소개한 α와 β의 경우 다양한 시나리오를 고려할 수 있도록 $\alpha\in\{0.3,0.5,0.7\}$, $\beta\in\{0.3,0.6,0.9\}$을 가정하였다. 또한 n, m, α, β의 각 조합에 대해서 10개의

<Table 1> Experimental Results: Comparison with MILP

n	α	β	$m=2$			$m=3$		
			B&B	MILP	Ratio	B&B	MILP	Ratio
10	0.3	0.3	0.01 (0)	0.29 (0)	116.8	0.03 (0)	0.82 (0)	55.9
		0.6	0.00 (0)	0.21 (0)		0.01 (0)	0.94 (0)	
		0.9	0.00 (0)	0.14 (0)		0.00 (0)	0.32 (0)	
	0.5	0.3	0.00 (0)	0.25 (0)		0.02 (0)	0.80 (0)	
		0.6	0.00 (0)	0.29 (0)		0.01 (0)	0.95 (0)	
		0.9	0.00 (0)	0.16 (0)		0.00 (0)	0.38 (0)	
	0.7	0.3	0.00 (0)	0.29 (0)		0.03 (0)	0.62 (0)	
		0.6	0.00 (0)	0.26 (0)		0.02 (0)	1.17 (0)	
		0.9	0.00 (0)	0.16 (0)		0.00 (0)	0.43 (0)	
20	0.3	0.3	2.84 (0)	1242.82 (0)	874.0	168.37 (0)	7443.93 (4)	50.4
		0.6	0.31 (0)	1567.79 (0)		61.45 (0)	10094.48 (8)	
		0.9	0.00 (0)	67.98 (0)		0.26 (0)	1639.03 (1)	
	0.5	0.3	3.53 (0)	1181.31 (0)		261.42 (0)	7232.52 (3)	
		0.6	0.47 (0)	1688.99 (0)		36.05 (0)	7871.85 (6)	
		0.9	0.01 (0)	51.05 (0)		0.69 (0)	2328.81 (0)	
	0.7	0.3	3.05 (0)	440.73 (0)		374.32 (0)	6560.99 (4)	
		0.6	1.34 (0)	3813.86 (0)		269.88 (0)	9029.81 (7)	
		0.9	0.01 (0)	50.24 (0)		5.88 (0)	7215.79 (5)	

인스턴스를 무작위로 생성하여 반복실험을 진행하였다. 실험이 무한정 길어지는 것을 방지하기 위해 분기한정 알고리즘과 혼합정수계획모형 모두 계산시간에 3시간이라는 제한을 두었다.

<Table 1>은 실험결과를 보여준다. <Table 1>의 4열과 5열은 각각 $m=2$인 경우에 대해 본 논문에서 제안된 분기한정 알고리즘과 Kim et al.[6]에서 제시된 혼합정수계획모형의 계산시간을 초단위로 보여준다. 앞서 언급한 바와 같이 각 조합에 대해 10개의 인스턴스를 무작위로 생성하였고 그 평균치를 기입하였다. 해당 열들에서 괄호안에 표시된 숫자는 총 10개의 인스턴스 중 3시간이라는 계산시간 제한 내에 최적해를 얻지 못한 경우의 수를 나타낸다. 6열은 4열과 5열에서 제시된 계산시간의 비율을 나타낸다. 예를 들어 116.8은 $n=10$, $m=2$인 경우 혼합정수계획모형의 평균 계산시간이 분기한정 알고리즘의 평균 계산시간의 116.8배임을 나타낸다. 7, 8, 9열은 $m=3$인 경우를 나타내며 그 의미는 각각 4, 5, 6열과 동일하다.

실험결과를 보면 모든 경우에 대해 본 논문에서 제안된 분기한정 알고리즘이 기존 방법에 비해 우수한 성능을 보임을 알 수 있다. 특히 $n=20$, $m=2$ 인 경우 기존 방법이 제안된 방법보다 최적해를 얻기까지 874배나 큰 계산시간이 필요함을 알 수 있다. 또한 $n=20$, $m=3$인 경우 제안된 분기한정 알고리즘의 경우 모든 경우에 대해 합리적인 시간 내에 최적해를 도출한 반면 혼합정수계획모형의 경우 3시간이라는 계산시간 제한 내에 최적해를 도출하지 못한 경우가 빈번히 발생하였다. 이러한 이유로 비록 <Table 1>에서 비율이 50.4로 계산되었지만 혼합정수계획모형의 경우 많은 경우에 계산시간 제한에 의해 중단되었기 때문에 실제 비율은 50.4보다 더 클 것이라고 예상할 수 있다. 이상의 결과를 통해 제안된 분기한정 알고리즘이 기존의 혼합정수계획모형에 비해 매우 우수한 성능을 보임을 확인할 수 있다.

n과 m이 커짐에 따라 문제의 복잡도가 높아지고 따라서 계산시간이 증가하는 것은 자명하며 <Table 1>에서도 쉽게 확인할 수 있다. 전술한 바와 같이 본 연구에서 제시한 분기한정 알고리즘이 혼합정수계획법에 비해 우수한 성능을 보이지만, 본 연구에서 다루는 문제의 복잡도가 NP-hard(in the strong sense)이므로 n과 m이 커짐에 따라 현실적인 계산시간 내에 해결할 수 있는 문제의 크기에 제한이 있다. 이러한 특성과 한계를 명확히 확인하기 위해, <Table 2>와 같이 추가실험을 수행하였으며, 이 때 α와 β값은 각각 0.5와 0.6으로 설정하였다. <Table 2>는 각각의 n과 m 조합에 대해 10개의 인스턴스를 무작위로 생성 후 분기한정 알고리즘을 적용(계산시간 제한: 3시간)하여 최적해를 얻기까지 소요된 계산시간을 초단위로 나타낸다. 이 때 앞선 실험과 유사하게 3시간 내에 최적해를 구하지 못한 경우 계산시간을 3시간(=10,800초)으로 기록하였으며, 괄호안의 숫자는 10개의 인스턴스 중 계산시간 제한 내에 최적해를 얻지 못한 인스턴스의 수를 나타낸다.

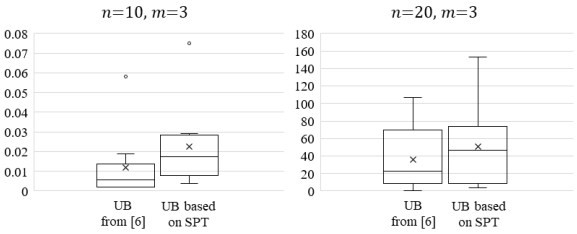
<Table 2> Experimental Results: Larger n and m

n	$m=2$	$m=3$	$m=4$	$m=5$
10	0.00(0)	0.01(0)	0.08(0)	0.54(0)
20	0.47(0)	36.05(0)	3764.70(2)	10800(10)
30	519.85(0)	9257.14(8)	10800(10)	-
40	6738.96(5)	10800(10)	-	-
50	9044.25(8)	-	-	-

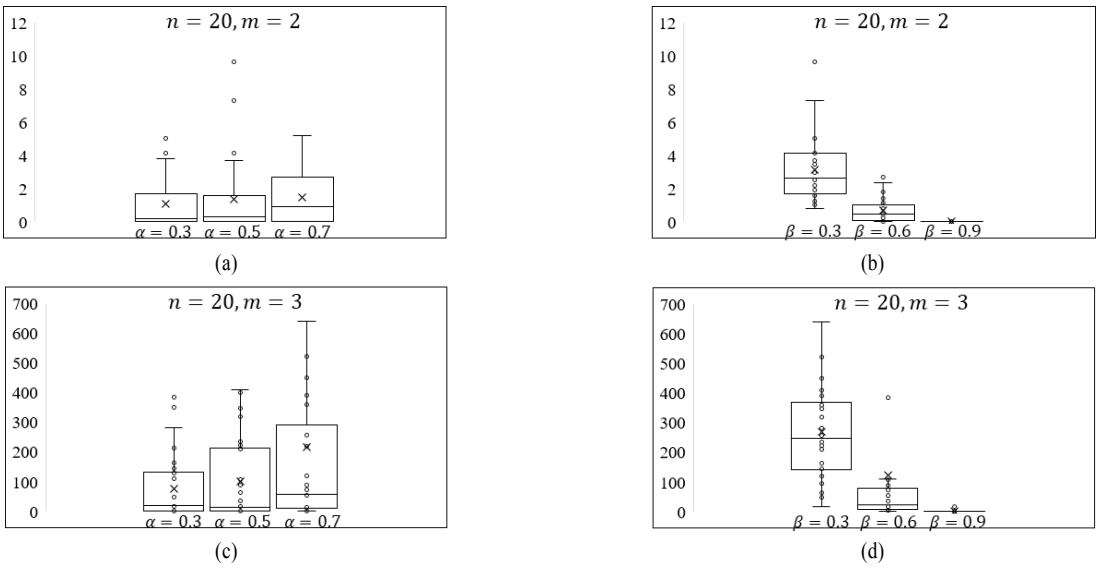
앞서 <Table 1>에서 실험이 이루어진 조합의 경우 해당 결과를 그대로 기록했으며, 더 작은 크기의 문제에 대해 10개 인스턴스 모두 계산시간 제한 내에 최적해가 도출되지 않은 경우 크기가 더 큰 문제에 대한 실험은 진행하지 않았다.

분기한정 알고리즘에서 핵심적인 역할을 하는 *LB*와 *UB*에 대한 추가 실험 또한 진행하였다. 본 연구에서 제안한 *LB*의 성능을 평가하기 위해 비교군으로 *LB*를 아예 사용하지 않는 경우와 앞서 2장에서 소개한 혼합정수계획모형의 정수형(integer) 결정변수 x_{ij}를 실수형 변수로 relaxation하여 선형계획법(linear programming, LP)으로부터 얻어지는 목적식 값을 *LB*로 사용하는 두 경우를 고려하였다. 두 대안 모두 $n \geq 20$, $m \geq 2$인 경우에 대해 3시간의 계산시간 내에 최적해를 도출하지 못하였다. 이는 본 논문에서 제안된 *LB*를 사용하면 $n=20$, $m=3$인 경우에도 평균 2분 정도의 계산시간으로 최적해를 얻을 수 있다는 점과 매우 대비되는 결과이며, 이를 통해 본 논문에서 제안된 *LB*의 우수성을 확인할 수 있다. *LB*를 아예 사용하지 않는 경우 계산시간이 매우 길어짐은 자명하며, LP-relaxation 기반 *LB*를 사용한 경우 또한 혼합정수계획법보다도 더 좋지 않은 성능을 보였는데, 이는 Gurobi와 같은 상용 소프트웨어들이 혼합정수계획모형을 풀기 위해 기본적으로 LP-relaxation 기반 *LB*를 사용하며 이에 추가로 효율적인 컷(cut)들을 사용하기 때문으로 해석할 수 있다. 결론적으로 제안된 *LB*는 본 연구에서 다루는 문제에 특화된 효율적인 방법이라고 평가할 수 있다.

앞서 소개한 바와 같이 초기 *UB*를 위해 Kim et al.[6]에서 제안된 휴리스틱 알고리즘을 활용하였다. 초기 *UB*의 성능에 따라 전체적인 분기한정 알고리즘의 계산시간이 영향을 받을 수 있으므로 그 효과를 평가하기 위한 추가 실험을 진행하였다. 비교군으로 쉽게 적용 가능한 최단작업시간우선 규칙 기반의 초기 *UB*를 고려하였다. <Figure 4>의 계산시간에 대한 상자수염그림(box plot)을 통해 확인할 수 있는 바와 같이 최단작업시간우선 규칙 기반의 초기 *UB*를 사용하는 경우 분기한정 알고리즘의 계산시간이 소폭 증가하지만 눈에 띄는 차이는 확인하기 어렵다 ($n \in \{10,20\}$, $m=3$, $\alpha=0.5$, $\beta=0.6$). 결론적으로 Kim et al.[6]에서 제안된 휴리스틱알고리즘을 활용하여 초기 *UB*를 결정하면 계산시간 감소에 긍정적인 부분이 있지만, 쉽게 활용 가능한 다른 대안을 적용해도 본 연구에서 제안한 분기한정 알고리즘을 통해 충분히 효율적으로 최적해를 도출할 수 있음을 확인할 수 있다.



<Figure 4> UB from [6] vs UB based on SPT



<Figure 5> Box plots of branch and bound algorithm computation times over α and β

마지막으로 파라미터 α와 β에 따라 문제의 특성이 어떻게 변화하는지를 살펴본다. <Table 1>에서 대략적인 경향을 확인할 수 있으나, 더욱 명확한 경향성을 관측하기 위해 $n=20$, $m\in\{2,3\}$인 경우 분기한정 알고리즘의 계산시간을 α와 β에 따라 상자수염그림으로 나타내면 <Figure 5>와 같다. $n=10$ 인 경우 계산시간 자체가 매우 작아 제외하였음을 밝힌다. <Figure 5>를 통하여 α와 β에 따른 계산시간 변화를 명확히 관찰할 수 있다. α가 커짐에 따라 즉, 구간별 작업시간 감소의 산포가 작을수록 문제의 복잡도가 높아지며 따라서 계산시간이 증가함을 확인할 수 있다. β의 경우 값이 작을수록, 즉, 구간의 길이가 짧아 한 구간에서 오직 소수의 작업만이 이루어질 때 문제의 복잡도가 높아지며 결과적으로 계산시간이 증가함을 확인할 수 있다.

5. 결론

본 연구에서는 복수의 중요시점이 존재하며 중요시점들을 기준으로 작업시간이 계단형으로 감소하는 계단형 향상 작업의 스케줄링 문제를 다루었다. 해당 문제의 최적해를 얻기 위해 분기한정 알고리즘을 개발하였으며, 실험을 통해 기존 방법 대비 제안된 방법이 우수한 성능을 보임을 확인하였다. 기존 방법 대비 계산 효율성을 크게 증가시켰지만, 본 연구에서 다루는 문제의 복잡도가 NP-hard이므로 문제의 크기가 커지면 여전히 합리적인 시간 내에 최적해를 도출할 수 없다는 한계가 존재한다. 따라서 향후 연구는 크게 두 가지 방향으로 진행될 수 있다고 판단된다. 첫째는 최적해 도출을 위해 추가적으로 계산 효율성을 향상시키는 방향이다. 부분 스케줄들을 비교하여 특정 스케줄이 다른 스케줄보다 항상 좋은 목적식 값을 도출함을 판단할 수 있는 dominance property의 개발, 본 연구에서 제시된 노드의 목적식 값 하한 보다 더욱 타이트한 하한의 개발 등을 생각해 볼 수 있다. 다른 방향으로는 최적해에 근사한 우수한 해를 빠른 시간 안에 찾을 수 있는 휴리스틱 방법의 개발을 생각해 볼 수 있다. 특히 본 연구에서 제안된 분기한정 알고리즘을 바탕으로 beam search 알고리즘 등을 적용한 휴리스틱 방법의 개발은 흥미로운 연구주제가 될 수 있다고 판단된다.

Acknowledgments

This work was supported by Chungnam National University.

References

[1] Berlinska, J., Scheduling for Data Gathering Networks with Data Compression, *European Journal of Operational Research*, 2015, Vol. 246, No. 3, pp. 744-749.

[2] Cheng, T. and Ding, Q., Single Machine Scheduling with Step-deteriorating Processing Times, *European Journal of Operational Research*, 2001, Vol. 134, No. 3, pp. 623-630.

[3] Cheng, T., He, Y., Hoogeveen, H., Ji, M., and Woeginger, G. J., Scheduling with Step-improving Processing Times, *Operations Research Letters*, 2006, Vol. 34, No. 1, pp. 37-40.

[4] Gawiejnowicz, S., A Review of Four Decades of Time-dependent Scheduling: Main Results, New Topics, and Open Problems, *Journal of Scheduling*, 2020, Vol. 23, No. 1, pp. 3-47.

[5] Ji, M., He, Y., and Cheng, T., A Simple Linear Time Algorithm for Scheduling with Step-improving Processing Times, *Computers & Operations Research*, 2007, Vol. 34, No. 8, pp. 2396-2402.

[6] Kim, E.-S., Choi, B.-C., Lee, J.-H., and Kim, H.-J., Scheduling of Step-improving Jobs with Multiple Critical dates, Technical Report, 2023.

[7] Kim, E.-S. and Oron, D., Minimizing Total Completion Time on a Single Machine with Step Improving Jobs, *Journal of the Operational Research Society*, 2015, Vol. 66, No. 9, pp. 1481-1490.

[8] Kim, H.-J., Kim, E.-S., and Lee, J.-H., Scheduling of Step-improving Jobs with an Identical Improving Rate, *Journal of the Operational Research Society*, 2022, Vol. 73, No. 5, pp. 1127-1136.

[9] Li, K., Chen, J., Fu, H., Jia, Z., and Wu, J., Parallel Machine Scheduling with Position-based Deterioration and Learning Effects in an Uncertain Manufacturing System, *Computers & Industrial Engineering*, 2020, Vol. 149, pp. 1-12.

[10] Luo, W., Gu, B., and Lin, G., Communication Scheduling in Data Gathering Networks of Heterogeneous Sensors with Data Compression: Algorithms and Empirical Experiments, *European Journal of Operational Research*, 2018, Vol. 271, No. 2, pp. 462-473.

[11] Sundararaghavan, P.S. and Kunnathur, A.S., Single Machine Scheduling with Start Time Dependent Processing Times: Some Solvable Cases, *European Journal of Operational Research*, 1994, Vol. 78, No. 3, pp. 394-403.

ORCID

Junho Lee | https://orcid.org/0000-0001-7693-8896