Journal of the Korean Institute of Industrial Engineers
Vol. 27, No. 2, pp. 158-168, June 2001.

작업 투입시점과 순서 의존적인 작업준비시간이 존재하는 단일 기계 일정계획 수립을 위한 Tabu Search

신현준[1†] · 김성식[1] · 고경석[2]

[1]고려대학교 산업시스템정보공학과 / [2]현대정보기술

A Tabu Search Algorithm for Single Machine Scheduling Problem with Job Release Times and Sequence-dependent Setup Times

Hyun Joon Shin[1] · Sung Shick Kim[1] · Kyoung Suk Ko[2]

[1]Department of Industrial Systems and Information Engineering, Korea University, Seoul, 136-701
[2]Information Technology R&D Center, Hyundai Information Technology, Yongin, 449-910

We present a tabu search (TS) algorithm to minimize maximum lateness on a single machine in the presence of sequence dependent setup times and dynamic job arrivals. The TS algorithm starts with a feasible schedule generated by a modified ATCS (Apparent Tardiness Cost with Setups) rule, then through a series of search steps it improves the initial schedule. Results of extensive computational experiments show that the TS algorithm significantly outperforms a well-known RHP heuristic by Ovacik and Uzsoy, both on the solutions quality and the computation time. The performance advantage is particularly pronounced when there is high competition among jobs for machine capacity.

Keywords : Sequence-dependent Setup Times, Release Times, Tabu Search, Single Machine, Scheduling

1. 연구의 배경 및 목적

본 연구에서는 납기 지연시간(Lateness Time)의 최대값(이하 L_{max})을 최소화하기 위해 N개의 작업(Job)을 단일기계(Single Machine)에 스케줄링하는 알고리듬을 제시한다. 각 작업의 인덱스를 $i(i = 1, \cdots, N)$라고 했을 때, 이들 작업은 각각 작업 투입시점(Release Time) r_i와 납기(Due Date) d_i를 갖는다. 각 작업의 가공시간(Processing Time) p_i는 작업들이 기계에서 처리되는 순서에 독립적으로 주어지지만, 작업 준비시간은 그 작업들이 가공되는 순서에 따라 달라지는 순서 의존적인 준비시간(Sequence-dependent Setup Time)을 갖는다. 여기서, 순서 의존적인 작업 준비시간이란 작업 i의 가공을 마친 후 작업 j를 준비하는 데 소요되는 시간을 s_{ij}라 할 때, $s_{ij} \neq s_{ji}$인 성질을 갖는 작업 준비시간을 의미한다. 알고리듬에 의해 생성된 스케줄 결과는 집합 $\Pi = \{\pi(1), \pi(2), \pi(3), \cdots, \pi(N)\}$으로 표현하며, 여기서 $\pi(j)$는 스케줄상의 j번째 위치해 있는 작업의 인덱스를 뜻한다. 본 연구의 일정계획 수립 목적은 L_{max}를 최소화하는 것으로 다음과 같이 표현될 수 있다.

$$Minimize\ L(\Pi) = \max\{T_i | i = 1, \cdots, N\}$$

이때, $T_i(T_i = \max(d_i - C_i, 0))$는 작업 i의 납기지연시간(Tardiness Time)이고, C_i는 작업 i의 가공완료시간(Completion Time)이다. 본 연구에서는 하나의 작업을 분할이 허용되지 않는 최소단위로 간주하며, 아직 도착하지 않은 작업을 위해 미리 해당 작업의 작업준비를 할 수 없다고 가정한다.

순서 의존적인 작업 준비시간은 반도체 공정, 자동차 도장공정 및 플라스틱 사출성형공장 등의 작업현장에 많이 존재하는데, 일반적으로 순서 의존적인 작업 준비시간이 존재하는 경우, 그 일정계획 수립 결과에 따라 자원의 가동률과 그에 따

† 연락저자 : 신현준, 136-701 서울시 성북구 안암동 1가 고려대학교 산업시스템정보공학과, Fax : 929-5888, e-mail : mrshin@korea.ac.kr
2001년 2월 접수, 1회 수정 후, 2001년 4월 게재 확정.

른 납기준수율이 크게 좌우된다. 특히, 반도체 공정과 같이 높은 설비 효율을 요구하는 생산환경에서는 효율적인 일정계획 수립 방법이 필수적이다. 더욱이, 근래의 소비자 주도 환경에서는 주문생산방식이 확산되고 기업 간 경쟁이 치열해지면서 납기 준수가 기업의 생존과 직결된 중요한 관리목표가 됨에 따라, 단일기계에서의 L_{max}를 최소화하는 문제(이하 L_{max} 문제)는 상당히 많은 응용분야가 존재하며, 실제로 L_{max} 문제는 생산현장에서 흔히 볼 수 있는 병렬기계(Parallel Machine)나 잡샵(Job Shop) 일정계획 문제의 하위 문제(Subproblem)로 사용되고 있다. 그러나 단일기계 일정계획 수립 문제에서 작업준비시간은 과거에도 자주 고려되었지만 작업의 투입가능시점이 존재하는 동적인 상황은 최근까지도 충분히 반영되지 않고 있다. 이로 인해 L_{max} 문제를 위해 개발된 단일기계 스케줄링 알고리듬을 작업 투입시점이 제약조건으로 요구되는 잡샵 등의 복잡한 문제로 확장하여 사용하는 데 어려움이 많았다.

따라서, 본 연구에서는 순서 의존적인 작업 준비시간과 작업 투입시점이 존재하는 동적인 환경에서의 L_{max} 문제를 위한 일정계획 수립 알고리듬을 제시한다. 제시된 알고리듬은 초기해를 구하는 알고리듬과 해를 개선하는 알고리듬으로 구성되어 있으며, 해를 개선하는 알고리듬은 메타 휴리스틱(Meta Heuristic) 방법 중의 하나인 타부 탐색(Tabu Search, 이하 TS)을 이용한다.

Ali *et al.* (1999)의 조사 논문을 보면 기존 연구 결과들이 자세히 정리되어 있다. 이들의 연구 결과들 중 본 연구와 동일한 문제에 대한 스케줄링 알고리듬을 제시한 기존 연구로는 Ovacik and Uzsoy (1994b)의 RHP(Rolling Horizon Procedure) 알고리듬이 있다. 본 연구에서는 제시된 알고리듬의 객관적인 성능 평가를 위하여 벤치마킹 데이터(Benchmarking Data)를 사용하여 RHP 알고리듬과 비교를 수행하고, 제시된 알고리듬이 수행속도 및 해의 성능 면에서 매우 우수한 결과를 나타낸다는 것을 보인다.

다음 2절에서는 기존 연구에 대하여 고찰하고 3절과 4절에서는 초기해를 생성하는 방법과 TS 알고리듬에 대하여 각각 설명한다. 5절에서는 실험을 통하여, 본 연구에서 제안한 알고리듬을 비교 대안 알고리듬과 비교 분석하고 그 성능을 평가한다. 마지막으로 6절에서는 본 연구의 결론과 추후 연구방향을 정리한다.

2. 기존 연구

작업 준비시간이 존재하지 않는 단일기계 일정계획 문제에서 L_{max}를 최소화하는 방법들은 지금까지 많이 연구되어 왔다. 단일기계 일정계획 문제는 작업 투입시점의 제약이 없을 경우 EDD(Earliest Due Date) 규칙을 이용하면 쉽게 최적해를 구할 수 있다(Baker, 1974). 그러나 작업 투입시점의 제약이 존재하는 문제($1|r_j|L_{max}$)는 NP-hard이며(Garey and Johnson, 1979), 특히 순서 의존적인 작업 준비시간이 함께 존재하는 본 연구의 문제($1|r_j, s_{ij}|L_{max}$)는 매우 NP-hard한 문제이다(Ovacik and Uzsoy, 1994).

Farn과 Muhlemann(1979)은 작업이 동적으로 투입되는 환경에서 순서 의존적인 작업 준비시간이 존재하는 문제를 연구하였고, 정적인 환경에서 가장 우수했던 발견적 기법(Heuristic)이 동적인 상황에서도 반드시 최상은 아니라는 것을 보여주었다. Bianco *et al.*(1988)은 $1|r_j, s_{ij}|C_{max}$ 문제를 혼합 정수 선형 계획법(Mixed Integer Linear Programming)의 형태로 나타내었고, 하한(Lower Bound)과 우세 기준(Dominance Criteria)을 이용한 발견적 알고리듬(Heuristic Algorithm)을 개발하였다.

Uzsoy *et al.*(1991)은 $1|prec, s_{ij}|L_{max}$ 문제에 대한 최적해를 찾기 위해 분지한계법(Branch-and-bound Algorithm)을 제시하였으나, 이 해법은 작업수가 대략 15개 이상으로 증가하면 수행 속도상의 큰 부담을 갖는다는 어려움이 있었다. Ovacik and Uzsoy (1994a)는 주어진 예측구간(Forecast Window) 내에서 가용한 작업들의 정보를 이용하여 $1|r_j, s_{ij}|L_{max}$ 문제를 풀었다. 그리고 Ovacik and Uzsoy(1994b)는 알고리듬의 수행속도상의 문제점을 해결하기 위하여 $1|r_j, s_{ij}|L_{max}$ 문제에 대해 일련의 RHP 알고리듬들을 제시하였으며, 이 알고리듬들이 근시안적인 할당규칙(Dispatching Rule)들보다 적절한 시간 내에서 매우 우수한 성능을 보인다는 것을 보였다.

Kim *et al.*(1995)은 납기지연 가중치의 합(Total Weighted Tardiness, 이하 TWT)을 최소화하는 $1|s_{ij}|TWT$ 문제를 풀기 위한 방안으로 할당규칙과 신경망(Neural Networks)을 결합한 혼합적인 방법(Hybrid Approach)을 사용하였다. Lee *et al.* (1997)은 역시 $1|s_{ij}|TWT$ 문제의 해법으로 ATCS(Apparent Tardiness Cost with Setups)규칙을 사용하는 3단계 발견적기법 절차를 제안하였고, 이것이 Raman *et al.*(1989)이 제안한 규칙보다 우수하다는 것을 보였다.

이상에서 살펴본 바와 같이 기존 연구들 중, Ovacik and Uzsoy(1994b)의 연구만이 본 연구의 문제($1|r_j, s_{ij}|L_{max}$)와 동일한 문제를 다루고 있다. 따라서 본 연구에서는 제안된 알고리듬의 우수성을 제5절에서 Ovacik and Uzsoy(1994b)가 실험한 데이터(Uzsoy)를 토대로 직접적인 비교 실험을 통해 분석하도록 한다.

3. 초기해 생성

본 절에서는 TS 알고리듬에 사용될 초기해를 생성하는 방법을 제시한다. 일반적으로 메타 휴리스틱을 이용한 탐색에 있어서 전체 탐색 영역 중, 최적지점에서 가까운 곳에서 출발하는 것이 결국 짧은 시간 내에 비교적 좋은 지점에 도달하게 된다고 알려져 있다(Crauwels *et al.*, 1996). 따라서, 본 연구에서

는 좋은 초기해로부터 타부 탐색을 수행하기 위한 방안으로 Lee *et al.*(1995)이 제안한 ATCS규칙을 수정하여 사용하는 MATCS(Modified ATCS)규칙을 새롭게 제안한다.

ATCS규칙은 순서 의존적인 작업 준비시간과 작업의 납기를 고려하여 납기지연 가중치의 합을 최소화하는 문제($1|s_{ij}|TWT$)를 풀기 위해 제안되었다. ATCS 규칙은 현재 스케줄 시점(t)에서 작업순서가 결정되지 않은 작업들(Ω) 중, 최근에 스케줄된 작업이 l일 때 그 다음 작업으로 인덱스($I_j(t, l)$)가 가장 큰 작업 $j(j = \{j \mid \max I_j(t, l),\ \text{for } j \in \Omega\})$가 스케줄 되도록 한다. 이 규칙을 적용함으로써 작업의 우선순위가 높을수록, 가공시간이 짧을수록, 납기까지의 여유가 없을수록, 그리고 순서 의존적인 작업 준비시간이 작을수록 그 해당 작업은 높은 인덱스를 갖게 되고 결과적으로 작업 순서상 앞 부분으로 스케줄 된다.

그러나 본 연구에서는 모든 작업이 작업 투입시점을 갖는다는 점, 작업의 우선순위가 고려되지 않는 점, 그리고 목적함수로 L_{max}를 사용한다는 점에서 ATCS 규칙을 그대로 적용하기 힘들다. 따라서, 본 연구에서는 ATCS 규칙의 인덱스 계산 수식을 수정하여 사용하는 MATCS규칙을 이용한다.

MATCS규칙에서 사용하는 인덱스 계산 수식은 수식 (1)과 같다.

수식 (1)에서, t는 작업 l의 가공 완료시점(C_l), $\bar{p}$와 $\bar{s}$는 각각 작업순서가 결정되지 않은 작업들의 평균 가공시간과 평균 작업 준비시간을 의미하고, k_1과 k_2는 조정모수(Scaling Parameter)로서 문제 특성에 맞게 주어지며 수식 (1)의 두번째 항과 세번째 항이 전체 인덱스 값에 미치는 영향을 조절해 주는 역할을 한다.

$$I_j(t, l) = \frac{1}{p_j} \exp\left(-\frac{d_j - p_j - t}{k_1 \bar{p}}\right) \exp\left(-\frac{s_{lj}'}{k_2 \bar{s}}\right) \quad (1)$$

$$\text{단, } s_{lj}' = \begin{cases} s_{lj} & \text{if } r_j \le t \\ r_j - t + s_{lj} & \text{otherwise} \end{cases}$$
$$j \in \Omega$$

수식 (1)의 두번째 항은 납기를 어긴 작업들에 대해 납기지연된 정도에 따라 1이상의 값을 차등 제공함으로써, 작업순서 결정시 납기지연 정도의 차이를 분별할 수 있게 한다. 세번째 항에서는 작업의 투입시점을 순서 의존적인 작업 준비시간인 s_{lj}에 반영하였는데, 만약 작업 j의 투입가능시점이 작업 l의 종료시점인 t보다 작거나 같다면, 즉 작업 j가 t시점에 투입가능(Available)하다면 s_{lj}'값으로 기존의 s_{lj} 값을 그대로 사용하고, 그와 반대로 작업 l의 종료시점과 작업 j의 가공시작시점 사이에 유휴시간(Idle Time)이 발생하게 된다면 s_{lj}' 값으로 기존의 s_{lj} 값에 유휴시간을 더한 값($= s_{lj} + r_j - t$)을 사용한다. 이렇게 함으로써, 불필요한 기계 유휴시간과 순서 의존적인 작업 준비시간을 함께 고려하는 스케줄 수립이 가능하다.

4. Tabu Search(TS)에 의한 해의 개선

MATCS를 이용한 초기해는 EDD 규칙과 같이 단순한 할당규칙보다는 좋은 해를 보여주지만, 할당규칙의 근시안적인 특성의 한계를 벗어나지 못하기 때문에 최적해에 근접한 해(Near optimal solution)를 보장해 주지 못한다. 따라서 본 연구에서는 초기해 생성단계에서 얻은 초기해를 TS를 통해 개선한다.

TS는 Glover(1989, 1990)에 의해 현재의 형태로 정립되었고, 시뮬레이티드 어닐링(Simulated Annealing)기법 및 유전 알고리듬(Genetic Algorithm)과 함께 현실적인 문제를 다루는 우수한 접근방법으로 알려져 있다. TS는 해를 탐색해 나가는 과정을 기억하여 중복탐색을 금지(Tabu)하고 해의 순환을 방지함으로써 지역최적해(Local Optimum)에서 머무르는 것을 방지해주므로, 조합최적화 문제에서 빠른 시간에 근사최적해를 찾는 데 적합한 기법이다. TS를 특정한 문제에 적용할 때, 해영역(Solution Space)에 대한 탐색성능은 초기해의 성능, 현재해(Current Solution)로부터 이웃해(Neighborhood Solution)들을 생성하여 다음해로 이동(Move)하는 방법, 이동에 관한 정보를 나타내는 타부 속성(Tabu Attribute), 그리고 타부 속성을 기억한 다음 얼마 후에 이동에 대한 타부상태를 해제할지를 결정하는 타부 목록(Tabu List) 크기 등에 의해 영향을 받는다(Glover, 1989, 1990).

일정계획 수립문제에 일반적으로 적용되는 TS 기법들과 비교해 볼 때, 본 연구에서 제안하는 TS 기법의 가장 큰 특징들 중의 하나는 탐색성능을 향상시키기 위해 고안한 이웃해 생성 및 이동방법이다. 일반적으로 일정계획 수립문제에 적용되는 TS는 현재해에서 두 작업들의 위치를 바꾸는 교환이동(Swap Move)이나 현재해의 한 작업을 다른 위치로 삽입하는 삽입이동(Insert Move)의 방법들을 통해 이웃해를 생성한다. 그러나 이러한 이동방법에 의해 현재해로부터 가능한 모든 이웃이 생성되고 평가된다면 작업의 수가 증가함에 따라 함께 증가하는 이웃해로 인해 계산시간이 너무 많이 소요된다. 따라서, 본 연구에서 제안하는 TS 기법은 각 작업들의 정보를 바탕으로 교환이동과 삽입이동, 그리고 교환이동과 삽입이동을 결합하여 사용하는 혼합이동(Hybrid Move) 방법에 대해 각각 제한된 이웃해 생성방법을 적용함으로써 탐색성능을 높이고 탐색에 소요되는 계산시간을 단축시킨다.

4.1 TS 알고리듬의 구조

본 연구에서 제안하는 TS의 이웃해는 앞서 언급한 이동방법 및 결정모수(Decision Parameter)인 탐색깊이(Search Depth)와 탐색반복횟수(Search Iteration Number)에 의해 결정된다. 이 결정모수들은 이웃해의 생성을 일정수준 이하로 제약함과 동시에 이웃해를 찾는 과정을 좀 더 집중적이면서 다양하게 수행할 수 있게 한다.

MATCS규칙에 의해 생성된 초기해 및 탐색과정에서 얻어

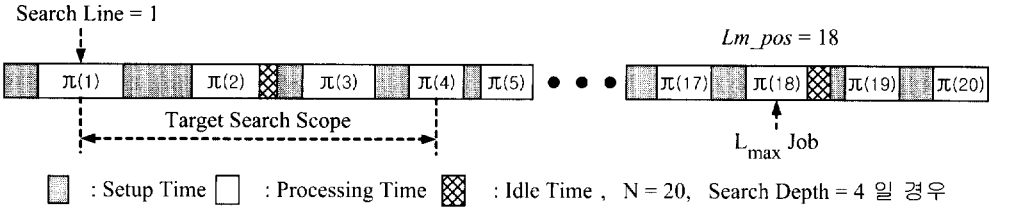

그림 1. 타부탐색 구조.

지는 이웃해에는 작업 투입시점과 납기, 그리고 순서 의존적인 작업 준비시간 등이 고려되어 있다. <그림 1>의 스케줄을 탐색과정에서 얻어진 이웃해 Π라 했을 때, 작업 순서상 선두에 위치해 있는 작업들은 대부분 납기가 급박하거나 작업 투입시점이 빠른 반면, 후미에 위치해 있는 작업들은 상대적으로 납기의 여유가 있거나 작업 투입시점이 늦는 특성을 갖는다. 그러므로, 작업 순서가 위치적으로 많은 차이를 보이는 모든 작업들을 동시에 현재해로부터 생성되는 이웃해의 대상으로 한다는 것은 탐색의 효율면에서 볼 때 의미가 없다. 이와 같은 문제 특성을 반영하여 스케줄상의 특정 작업의 위치를 탐색기준(Search Line)으로 하고 그 탐색기준으로부터 탐색깊이 범위 안의 작업들을 탐색영역(Target Search Scope)으로 설정함으로써, 현재해로부터 생성되는 이웃해를 탐색영역 내로 제한한다. 여기서, 탐색깊이는 생성되는 이웃해의 수에 직접적인 영향을 미치는 결정모수이며 주어진 탐색깊이에 따른 수행속도와 해의 성능에 대한 분석은 제5절에 제시되어있다.

<그림 1>에서 탐색기준은 $\pi(1)$작업의 위치에 해당하는 1로 설정되어 있으며, 이 탐색기준의 값은 변수로서 탐색영역 내의 작업들에 대한 탐색이 완료되고 난 후 1씩 증가하게 된다. 그리고 탐색깊이는 4로 설정되어 있는데 이 값은 알고리듬 수행 전에 주어지는 상수값으로 탐색기준이 가리키는 작업을 포함해서 이 값까지에 위치한 작업들을 탐색영역으로 지정하게 된다. <그림 1>의 경우 $\pi(1)$, $\pi(2)$, $\pi(3)$, $\pi(4)$의 4개의 작업들이 탐색영역이 되면서 이웃해 생성의 대상이 된다.

납기지연이 L_{max} 인 작업의 위치를 Lm_pos 라고 했을 때, <그림 2>의 경우 L_{max} 작업이 $\pi(18)$이므로 Lm_pos 값은 18이 된다. 이 Lm_pos는 이동에 의해 현재해가 바뀌고 그로 인해 L_{max} 작업이 변동될 경우 함께 변경되는 변수다. 탐색기준은 Lm_pos 값을 넘을 수 없는데, 그 이유는 Lm_pos 이후에 위치한 작업들을 탐색영역에 포함시켜 탐색하는 것은 목적함수 값을 감소시키는 데 긍정적인 영향을 주지 않기 때문이다.

<그림 2>는 탐색기준이 Lm_pos에 근접해 있을 경우에 설정되는 탐색영역의 범위를 보여주고 있다. 탐색기준이 16이고 탐색깊이가 4일 때, 원칙적으로 탐색영역은 $\pi(16)$, $\pi(17)$, $\pi(18)$, $\pi(19)$작업들로 설정되지만 Lm_pos 값이 18이므로, $\pi(19)$작업은 탐색영역의 범위에서 제외된다. 마찬가지로, Lm_pos가 17이고 탐색기준이 16, 탐색깊이가 4라면, 탐색영역은 $\pi(16)$, $\pi(17)$작업들로 설정된다. 자세한 TS 알고리듬의 절차는 다음과 같다.

[TS 알고리듬 절차]

Step 0. MATCS를 이용하여 초기해 Π와 목적함수값 $L(\Pi)$을 구하고, 이것을 각각 현재해 Π_{cur}와 $L(\Pi_{cur})$ 및 최우수해(Best Schedule) Π_{Best}와 $L(\Pi_{Best})$로 설정한다.

Step 1. 탐색기준과 현재 탐색횟수(Current Search Number)값을 1로 각각 설정한다. 탐색깊이와 탐색반복횟수를 각각 주어진 값으로 초기화하고, 삽입이동, 교환이동, 혼합이동 중 하나의 이동방법을 선택한다.

Step 2. Π_{cur}의 탐색영역 내에 있는 작업들에 대해 Step 1에서 선택한 이동방법의 이웃해 생성규칙을 이용하여, 가능한 모든 이웃해 $N(\Pi_{cur})$을 생성하고 생성된 이웃해들의 목적함수 값을 계산한다.

Step 3. $N(\Pi_{cur})$ 중 목적함수 값이 가장 작은 이웃해 Π^*를 현재해 Π_{cur}로 바꾸고 이때의 이동을 해당 이동방법의 타부 목록에 저장해 둔다. 만약, $L(\Pi^*)$ 값이 $L(\Pi_{Best})$ 값보다 작다면 Π_{Best}와 $L(\Pi_{Best})$을 각각 Π^*과 $L(\Pi^*)$로 치환한다.

Step 4. 탐색기준을 1증가시키고, Step 2로 되돌아간다. 이때,

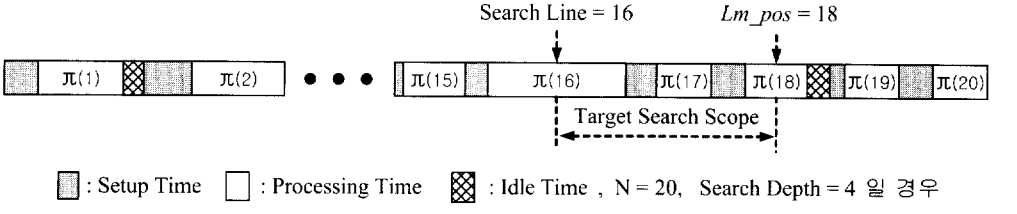

그림 2. 탐색영역 범위 설정.

만약 증가시킨 탐색기준이 Lm_pos 값과 일치한다면 탐색기준을 1로 초기화시키고 현재 탐색횟수값을 1 증가시킨다. 그리고 만약 증가시킨 현재 탐색횟수값이 미리 설정된 탐색 반복횟수값보다 크다면 타부탐색을 종료시킨다.

Step 2와 Step3에서는 삽입이동, 교환이동, 또는 혼합이동 중 선택된 이동방법을 이용하여 이웃해 집합 $N(\Pi_{cur})$을 생성한 후, 그 중 가장 좋은 해 Π^*로 현재해를 이동시키고 이동의 원인이 된 작업교환 정보를 해당 이동방법의 타부 목록에 저장한다. 그리고 지금까지 구했던 가장 좋은 스케줄 Π_{Best}와 비교하여 $L(\Pi^*)$이 $L(\Pi_{Best})$보다 우수하다면 Π_{Best}을 Π^*로 갱신해 준다. Step 4에서는 탐색기준을 1증가시키고 Step 2로 간다. 이때, 증가된 탐색기준 값이 Lm_pos 보다 클 경우 1회의 탐색이 종료되고 현재 탐색횟수를 1증가시킨 후, Step 2로 되돌아간다. 종료조건으로는 Step 4에서 현재 탐색횟수가 미리 설정된 탐색 반복횟수에 도달하게 되면 타부탐색은 종료되고, 현시점의 Π_{Best}가 최종해가 된다.

4.2 TS의 이웃해 생성 방법

본 연구에서는 삽입이동, 교환이동, 그리고 혼합이동의 3가지 방법에 의해 이웃해를 생성한다. 그 각각의 방법에 따른 이웃해 생성 방법은 다음과 같다.

4.2.1 삽입이동에 의한 이웃해 생성 방법

삽입이동은 한 작업을 선택하여 스케줄상의 다른 위치에 삽입하여 이웃해를 생성해 내는 방법이다. 삽입이동에 의해 이웃해를 생성해 내기 위해서는 옮길 작업과 옮겨갈 위치를 결정해 주어야 한다. 옮길 작업은 탐색 영역 내의 모든 작업이 해당되고, 옮겨갈 위치는 다음의 삽입이동 조건을 만족하는, 탐색 영역을 포함한 모든 위치이다.

• 삽입이동 조건

옮길 작업의 작업순서를 j라하고 옮겨갈 위치를 k라 할 때,

i) $j \neq k$

ii) $r_{\pi(j)} \leq C_{\pi(k)}$ for $j > k$

iii) 작업 $\pi(j)$가 위치 k로 이동하였을 때의 예상 종료시점 $(C_{\pi(j)}) \leq d_{\pi(j)} + L(\Pi_{cur})$. for $j < k$

iv) 타부 목록에 의해 금지된 이동이 아님.

예를 들어, <그림 2>의 경우 탐색영역 내의 작업들인 $\pi(16)$, $\pi(17)$, $\pi(18)$이 옮길 작업이 된다. 옮길 작업이 $\pi(16)$이라면 옮겨갈 위치는 작업순서 1부터 20까지의 위치 중, 위의 삽입이동 조건을 만족시키는 곳들이 해당된다. 따라서 작업수가 N일 경우, 일단 옮길 작업이 결정되어 옮겨갈 위치의 수가 $N-1$이라고 했을 때 한번의 이동을 위해 생성되고 평가받는 이웃해의 수는 최대 $(N-1)\times$(탐색깊이)를 넘지 않는다. 또한, 탐색기준이 Lm_pos에 가까워짐에 따라 탐색영역의 범위가 작아진다는 점과 함께 삽입이동 조건이라는 제약으로 인해 실제로 생성되는 이웃해의 수는 실험적인 경험에 의하면 대부분 위의 최대 이웃해 생성수를 크게 밑돌게 된다. 이것은 전형적인 삽입이동의 경우 한번의 이동을 위해 생성되는 이웃해의 수가 $N\times(N-1)$이라는 것과 비교해 볼 때, 적절한 탐색깊이 설정에 따라 이웃해가 상당한 수로 감소된다는 것을 알 수 있다.

TS 알고리듬 절차 중, 삽입이동의 역할은 Step 2에 해당되며 현재해 Π로부터 최대 $(N-1)\times$(탐색영역의 범위)개의 이웃해 $N(\Pi_{cur})$을 생성, 평가하여 $N(\Pi_{cur})$ 중 가장 우수한 해 Π^*를 구해내는 것이다.

4.2.2 교환이동에 의한 이웃해 생성 방법

교환이동은 두 작업의 작업 순서를 서로 교환함으로써 이웃해를 생성하는 방법이다. 교환이동에 의해 이웃해를 생성해 내기 위해서는 작업 순서를 교환할 서로 다른 두 개의 작업을 선택해 주어야 하는데, 이 두 작업은 탐색영역 내에 있으면서 다음의 교환이동 조건을 만족하는 모든 작업의 조합으로 이루어진다.

• 교환이동 조건

교환할 두 작업의 위치를 각각 j, k라 할 때,

i) $r_{\pi(j)} \leq C_{\pi(k)}$ for $j > k$

ii) 타부 목록에 의해 금지된 이동이 아님.

<그림 2>의 경우 탐색영역 내에 존재하는 작업들은 $\pi(16)$, $\pi(17)$, $\pi(18)$이고, 이 작업들을 대상으로 형성되는 조합 중 위의 교환이동 조건을 만족시키는 작업들에 대해서만 이웃해가 생성된다. 따라서, 작업수가 N일 경우, 한번의 이동을 위해 생성되고 평가받는 이웃해의 수는 최대 (탐색깊이)×(탐색깊이 − 1)/2를 넘지 않는다. 이것은 전형적인 교환이동의 경우, 한번의 이동을 위해 생성되는 이웃해의 수가 $N\times(N-1)/2$라는 것과 비교해 볼 때, 본 연구에서 제안하는 삽입이동 방법은 탐색영역이 적정 수준으로 설정되었을 때 매우 적은 수의 이웃해를 생성한다는 것을 알 수 있다.

4.2.3 혼합이동에 의한 이웃해 생성 방법

혼합이동은 현재 탐색횟수가 증가할 때마다 삽입 이동과 교환 이동을 서로 교환해 가면서 사용하는 방법이다. 즉, 타부탐색 알고리듬의 절차 중 Step 4에서 탐색기준 값이 Lm_pos보다 커지면 다시 탐색기준의 값이 1이 되면서 한번의 탐색을 마감하고 다시 다음 번 반복에 해당하는 새로운 탐색을 시작하게 되는데, 이때 이동방법을 서로 바꾸어준다. 그리고 각 반복마다 사용되는 삽입이동과 교환이동 방법은 앞서 기술한 내용과 동일하다.

4.3 TS의 타부 목록(Tabu List) 관리 방법

TS에서 타부 목록은 이동의 순환을 방지하는 역할을 한다. TS가 적용된 기존 연구들에서 타부 목록 크기는 대체적으로 6에서 10사이의 값이 사용됐는데(Laguna *et al.*, 1991, 1993), 본 연구에서는 이러한 기존 연구와 실험 경험을 바탕으로 타부 목록 크기를 7로 고정하여 사용하도록 한다.

4.3.1 삽입이동의 타부 목록 관리 방법

삽입이동에서는 옮길 작업의 인덱스와 옮길 작업 앞뒤에 위치한 작업의 인덱스를 타부 목록에 기억해 둘 속성으로 정하였다. 즉, 최근의 이동에서 옮길 작업이 $\pi(i)$이었다면 작업 $\pi(i)$ 및 선후 작업들의 인덱스들을 $\{\pi(i-1),\ \pi(i),\ \pi(i+1)\}$ 형식의 속성으로 기억해 두어, 타부 목록 크기 동안의 이웃해 생성시 작업 $\pi(i)$가 작업순서상 작업 $\pi(i-1)$과 작업 $\pi(i+1)$의 위치 사이로 되돌아오는 경우를 금지한다. 만약, 옮길 작업이 작업 순서상 첫번째이거나 마지막 위치인 $\pi(1)$나 $\pi(N)$라면 각각 {null, $\pi(1)$, $\pi(2)$}와 {$\pi(N-1)$, $\pi(N)$, null}의 속성이 타부 목록에 저장된다.

4.3.2 교환이동의 타부 목록 관리 방법

교환이동에서는 교환이동의 대상인 두 작업들의 인덱스를 타부 속성으로 타부 목록에 기억한다. 즉, 최근의 교환이동에서 작업 $\pi(i)$와 작업 $\pi(j)$를 교환하였다면 $\{\pi(i),\ \pi(j)\}$를 타부 속성으로 기억해 두고 타부 목록 크기 동안의 이웃해 생성시 작업 $\pi(i)$와 작업 $\pi(j)$를 교환하는 이웃해는 생성하지 않음으로써 해의 순환을 막는다.

4.3.3 혼합이동의 타부 목록 관리 방법

혼합이동의 경우에는 삽입이동과 교환이동에서 사용되는 각각의 타부 목록을 동시에 유지한다. 즉, 현재 사용되고 있는 이동방법이 삽입이동이라면 삽입이동의 타부 목록을, 그 반대로 현재 사용되는 이동방법이 교환이동이라면 교환이동의 타부 목록을 사용한다. 그러나 이 두 개의 타부 목록이 함께 사용되기 위해서는 특정 이동방법이 사용될 때, 해당 이동방법에서 기억한 타부 속성을 다른 이동방법이 탐색을 수행할 때 반영해 주어야 하는데 이러한 동기화 작업을 위해서 고려해야 할 사항은 다음과 같다.

① 인접해 있는 두 작업을 삽입/교환이동하였을 경우 :
일반적으로 삽입이동과 교환이동에서 기억되는 타부 속성은 서로의 이동을 제약하는 데 영향을 미치지 않지만 인접해 있는 두 작업을 이동하였을 때, 이 타부 속성을 두 개의 타부 목록에 기억시켜 동기화하지 않으면 <그림 3>과 같이 해의 순환이 발생할 수 있다. 따라서, 현재 이동방법이 교환이동일 경우 작업 $\pi(i)$와 작업 $\pi(i+1)$이 교환되어 이동된다면, 교환이동의 타부 목록에는 $\{\pi(i),\ \pi(i+1)\}$의 속성을 저장하고 삽입이동의 타부 목록에는 $\{\pi(i-1),\ \pi(i),\ \pi(i+1)\}$의 속성을 함께 저장한다.

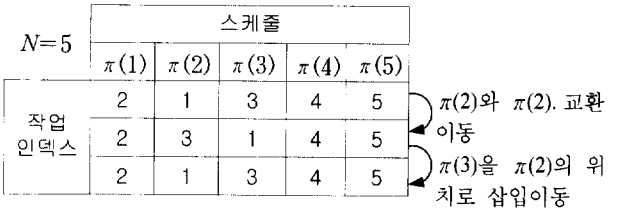

그림 3. 혼합이동에서의 해의 순환 예.

② 인접해 있지 않은 두 작업을 삽입/교환이동하였을 경우 :
위 ①의 경우를 제외하면 삽입이동과 교환이동에서 기억되는 타부 속성은 서로의 이동을 제약하지 않는다. 그러나 삽입이동과 교환이동의 타부 목록 크기를 각각 7로 사용한다면 이웃해 생성에 대한 제약이 너무 강해지므로, 전체적으로 이 두 개의 타부 목록의 크기를 7로 동기화하여 사용할 필요가 있다. 따라서, 현재 이동방법에 해당하는 타부 목록에는 해당 이동에 따른 속성을 저장하고 나머지 다른 이동방법의 타부 목록에는 널(Null) 속성을 저장한다. 즉, 현재 이동방법이 교환이동일 경우 작업 $\pi(i)$와 작업 $\pi(i+3)$이 교환되어 이동된다면, 교환이동의 타부 목록에는 $\{\pi(i),\ \pi(i+3)\}$의 속성을 저장하고 삽입이동의 타부 목록에는 널 속성인 {}을 동시에 저장한다.

4.4 열망 수준(Aspiration Level)

열망 수준이란 현재해 Π로부터 생성된 특정 이웃해가 타부에 의해 금지되고 있는 해일지라도 더 나은 해를 찾기 위하여 선택할 수 있는 수준을 의미한다. 본 연구에서는 열망 수준을 현 시점까지 구해진 최적해 Π_{Best}의 목적함수 값 $L(\Pi_{Best})$로 설정하였다.

5. 비교 대안 및 실험

5.1 비교 대안 및 실험 데이터

비교 대안으로는 제2절에서 언급했던 Ovacik and Uzsoy (1994b)의 RHP 알고리듬을 사용한다. RHP 알고리듬은 전체 계획구간(Planning Horizon) 내에 존재하는 작업들을 일정한 기간의 예측구간(Forecast Window)으로 분해(Decomposition)하여 하위문제를 구성하고, 구성된 하위문제를 대상으로 이미 설정된 값인 최대 k개의 작업에 대해 분지한계법을 이용하여 최적해를 구한다. 그리고 구해진 최대 k개의 작업으로 구성된 부분 최적해 중 역시 미리 주어진 값인 λ개만큼의 작업들만 전체 스케줄로 확정하여 포함시키는 일련의 과정을 반복하는

절차로 구성되어 있다.

Ovacik and Uzsoy(1994b)는 k와 λ 등의 결정모수들의 조합으로 이루어지는 72개의 알고리듬 조합과 함께 EDD규칙과 이를 향상시킨 최소 납기우선-국지개선(EDD-LI)규칙을 포함하여 모두 74가지의 방법들을 이용하여 해를 구하였고, 이때 사용된 실험 데이터와 74가지의 방법들을 이용하여 얻은 해 중 가장 우수한 결과(이하 O&U해)를 그들의 웹사이트(Uzsoy)에 제시하였다. 따라서, 본 연구에서는 제시된 알고리듬과 RHP 알고리듬의 성능을 그들이 사용한 실험 데이터와 결과를 토대로 직접적으로 비교 분석하도록 한다.

실험에 사용된 데이터의 생성 기준은 다음과 같다. 가공시간과 작업 준비시간은 각각 [1, 200]의 범위를 갖는 균일분포(Uniform distribution)로부터, 작업 투입시점은 0으로부터 평균 최대 완료시간(Expected Makespan : T)의 R 배까지인 [0, $R*T$]의 범위를 갖는 균일분포로부터 생성된다. 여기서, T는 평균 작업 준비시간(Mean Setup Time)과 평균 가공시간(Mean Processing Time)의 합과 작업수 N의 곱으로 계산되며, R은 작업 투입시점의 범위 모수(Release Time Range Parameter)로서 본 실험에서는 0.6, 0.8, 1.0, 1.2, 1.4의 값이 사용된다. R의 값이 작을수록 작업이 빈번하게 도착하여 기계 앞에서 가공을 대기하는 작업수가 증가하게되고, 반대로 R의 값이 클수록 작업이 드문드문 도착하기 때문에 상대적으로 기계 앞에서 대기하는 작업수가 줄어들게 된다. 또한, R 값이 0이라면 해당 문제는 정적인 문제(Static Problem)가 되며 모든 작업이 스케줄 시점에 가용하다. 납기는 식 (2)에 의해 생성되며, 수식에 사용된 α 값은 [-1, 4]의 범위를 갖는 균일분포로부터 발생된다.

$$d_i = r_i + 2\alpha p_i \tag{2}$$

본 실험에서 사용하는 벤치마킹 데이터는 <표 1>과 같이 R 값과 작업수의 조합으로 생성된 1000개의 문제로 구성되어 있다. 본 연구에서 제시한 알고리듬은 모두 C++를 이용하여 구현하였고, 일반 데스크탑 컴퓨터에서 실험하였으며 이때 사용된 CPU는 Pentium Ⅱ 400이다.

5.2 초기 실험

본 연구에서 제안된 알고리듬들을 통하여 얻는 결과는 결정모수인 탐색깊이와 탐색 반복횟수에 의하여 결정되고, 그 정도는 초기해의 질(Quality) 및 제시된 세 가지 이웃해 생성 방법에 따라 차이를 보인다. 본 절에서는 MATCS규칙과 기존의 할당규칙들과의 수행 성능을 비교하고, MATCS규칙을 이용하여 얻은 초기해가 전체 알고리듬의 결과에 미치는 영향에 대하여 평가한다. 또한, 탐색 반복횟수가 알고리듬 수행시간 및 해의 질에 미치는 영향을 실험을 통하여 분석하여 효율적인 탐색 반복횟수값을 결정한다.

초기 실험을 위한 데이터로는 앞서 언급한 문제집합(Problem Set) 중 작업수가 100인 문제들을 이용하고 결정모수인 탐색깊이는 100으로 고정하여 실험한다.

표 1. 실험 데이터 생성 기준

	사용 값	합계
작업 투입시점 범위 모수(R)	0.6, 0.8, 1.0, 1.2, 1.4	5
작업수	10, 20, 30, …, 100	10
조합수		50
조합당 문제수		20
전체 문제수		1000

5.2.1 초기해의 성능 평가 및 탐색횟수의 결정

본 실험에서는 제시된 알고리듬으로 실험하여 얻은 L_{max} 값을 O&U해로 나누어 준 비교값(Comparison Value)을 해의 질을 가늠하는 척도로 사용한다. 즉, 비교값이 1보다 작을 경우 제시한 알고리듬의 결과가 O&U해보다 좋다는 것을, 반대로 1보다 클 경우 좋지 않다는 것을 의미한다.

$$\text{비교값} = \frac{\text{TS 알고리듬을 이용하여 얻은 스케줄의 } L_{max}}{\text{RHP 알고리듬을 이용하여 얻은 스케줄의 } L_{max}(\text{O\&U 해})} \tag{3}$$

MATCS규칙, ATCS규칙, EDD규칙, 그리고 ER규칙을 사용하여 얻은 초기해의 질과 수행속도를 비교한 결과는 <그림 4>와 같다.

MATCS규칙이 근소한 속도차이를 보이면서 나머지 세 개의 할당규칙에 비해 O&U해에 근접해 있다는 것을 알 수 있다. 그리고 10에서 60까지 주어진 6개의 탐색 반복횟수값에 따라 위의 네 개의 할당규칙을 초기해 생성규칙으로 사용하여 수행된 TS 알고리듬 결과는 <그림 5>에 나타나있다. 여기서, <그림 5>의 TS-MATCS는 MATCS규칙을 초기해 생성규칙으로 사용한 TS 알고리듬을 의미하며, 이때의 비교값은 혼합이동 방법을 사용한 TS 알고리듬(이하 TS-Hybrid)과 삽입이동 방법을 사용한 TS 알고리듬(이하 TS-Insert), 그리고 교환이동 방법을

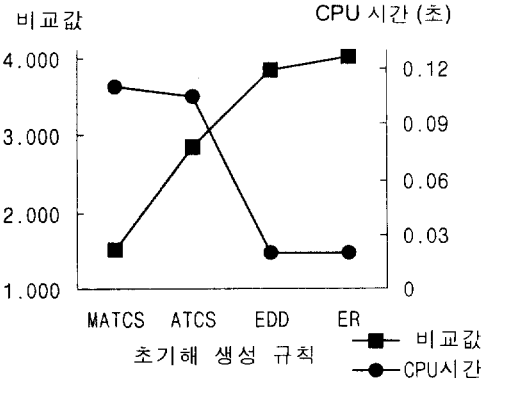

그림 4. 초기해 생성규칙간 성능 비교.

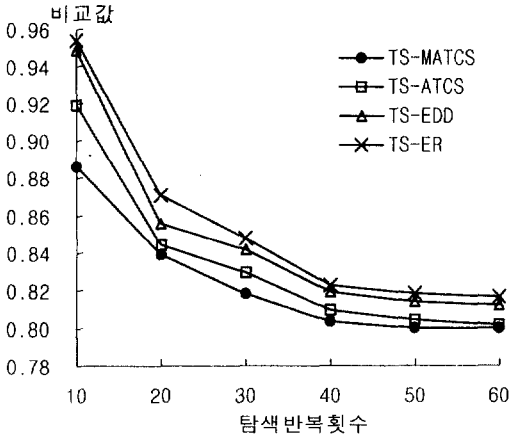

그림 5. 탐색 반복횟수에 따른 성능 비교.

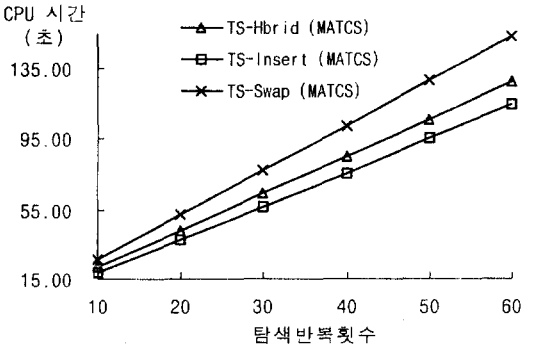

그림 6. 탐색 반복횟수에 따른 수행 속도.

사용한 TS 알고리듬(이하 TS-Swap)을 사용하여 얻은 비교값들의 평균이다. <그림 4>와 <그림 5>에서 알 수 있듯이 네 개의 할당규칙을 사용하여 얻은 초기해 값은 큰 차이를 보이지만 탐색 반복횟수가 늘어날수록 그 차이가 줄어들어 특정 값들로 수렴해 가는 것을 볼 수 있다. 그러나 현실적으로 적은 수행시간을 요하는 문제에서는 10 이하의 적은 탐색 반복횟수값으로 TS-MATCS 알고리듬을 사용한다면 해의 질과 수행속도 측면에서 효과적일 수 있다.

<그림 6>은 탐색 반복횟수값에 따른 TS 알고리듬의 수행속도 변화를 보여준다. 이 결과를 보면 탐색 반복횟수가 증가함에 따라 수행속도가 거의 선형적으로 비례하여 증가한다는 것을 알 수 있다. 또한, <그림 5>는 탐색 반복횟수가 증가할수록 해가 향상되지만 탐색 반복횟수가 일정한 값에 도달하게 되면 해의 향상이 거의 없다는 것을 보여주고 있다. 따라서, 본 실험에서는 알고리듬 수행속도 및 해의 질을 고려하여 40회를 효율적인 탐색 반복횟수값으로 결정하도록 한다.

5.3 실험 결과 및 분석

5.1에서 언급한 모두 1,000개의 실험 데이터를 사용하여 본 연구에서 제시한 TS 알고리듬의 성능을 O&U해와 비교해 보기로 한다. 초기 실험 결과 초기해 생성규칙은 MATCS규칙을

표 2. 실험 계획

결정모수	사용 값
초기해 생성규칙	MATCS규칙
탐색 반복횟수	40
탐색깊이	10, 20, 30, … , 90, 100
이웃해 생성 방법	TS-Hybrid, TS-Insert, TS-Swap

사용하고 탐색 반복횟수는 40회로 고정한다. 그리고 다른 하나의 결정모수인 탐색깊이와 세 개의 이웃해 생성 방법이 TS 알고리듬의 성능에 미치는 영향을 분석하기 위하여 <표 2>와 같이 실험을 계획한다.

탐색깊이를 10단위로 10에서 100까지 변화시켜 가며 작업투입시점 범위 모수인 R 값과 세 가지 이동방법, 그리고 작업수 N에 따른 TS 알고리듬의 실험 결과는 <표 3>에 정리되어 있고, 이 표에 사용된 결과값은 모두 비교값이다. <표 3>에서 Avg(R, *)은 특정 R 값이 주어졌을 때 모든 작업수 N에 대한 해(비교값)의 평균을, Avg(*, N)은 특정 작업수 N이 주어졌을 때 모든 R 값에 대한 비교값의 평균을, 그리고 Avg(*, *)은

표 3. R 값 및 탐색깊이와 이동방법 따른 TS 알고리듬 비교값

TS-Hybrid

탐색깊이	10	20	30	40	50	60	70	80	90	100
Avg(0.6, *)	1.078	0.917	0.855	0.833	0.818	0.809	0.806	0.808	0.805	0.805
Avg(0.8, *)	1.173	0.979	0.924	0.903	0.901	0.894	0.891	0.889	0.888	0.888
Avg(1.0, *)	1.193	1.007	0.967	0.950	0.946	0.947	0.943	0.945	0.943	0.942
Avg(1.2, *)	1.187	1.028	0.997	0.989	0.987	0.987	0.989	0.988	0.987	0.988
Avg(1.4, *)	1.220	1.037	1.006	1.000	0.997	0.995	0.998	0.999	0.998	0.998
Avg(*, *)	1.17	0.992	0.950	0.935	0.930	0.926	0.925	0.926	0.924	0.924
Avg(*, 10)	1.021	–	–	–	–	–	–	–	–	–
Avg(*, 20)	0.997	0.988	–	–	–	–	–	–	–	–
Avg(*, 30)	0.996	0.947	0.947	–	–	–	–	–	–	–
Avg(*, 40)	1.015	0.945	0.924	0.926	–	–	–	–	–	–
Avg(*, 50)	1.026	0.932	0.905	0.895	0.897	–	–	–	–	–
Avg(*, 60)	1.110	0.938	0.902	0.889	0.885	0.885	–	–	–	–
Avg(*, 70)	1.149	0.960	0.896	0.873	0.868	0.862	0.856	–	–	–
Avg(*, 80)	1.229	0.971	0.908	0.882	0.868	0.866	0.860	0.862	–	–
Avg(*, 90)	1.252	0.972	0.905	0.867	0.853	0.841	0.840	0.840	0.834	–
Avg(*, 100)	1.256	0.994	0.894	0.868	0.847	0.837	0.824	0.827	0.823	0.820

TS-Insert

Avg(0.6, *)	0.924	0.837	0.814	0.816	0.814	0.820	0.821	0.816	0.813	0.811
Avg(0.8, *)	1.022	0.924	0.914	0.913	0.904	0.917	0.915	0.916	0.918	0.921
Avg(1.0, *)	1.092	1.001	0.987	0.994	0.983	0.989	0.985	0.979	0.972	0.977
Avg(1.2, *)	1.163	1.068	1.028	1.034	1.021	1.014	1.015	1.020	1.014	1.015
Avg(1.4, *)	1.184	1.062	1.062	1.062	1.028	1.033	1.023	1.022	1.024	1.024
Avg(*, *)	1.077	0.983	0.961	0.964	0.950	0.954	0.952	0.951	0.949	0.950
Avg(*, 10)	1.022	–	–	–	–	–	–	–	–	–
Avg(*, 20)	1.010	1.021	–	–	–	–	–	–	–	–
Avg(*, 30)	0.970	0.965	0.972	–	–	–	–	–	–	–
Avg(*, 40)	0.980	0.957	0.948	0.963	–	–	–	–	–	–
Avg(*, 50)	0.957	0.922	0.924	0.921	0.917	–	–	–	–	–
Avg(*, 60)	0.972	0.905	0.895	0.903	0.906	0.906	–	–	–	–
Avg(*, 70)	1.002	0.906	0.881	0.875	0.883	0.892	0.883	–	–	–
Avg(*, 80)	1.036	0.921	0.896	0.895	0.872	0.881	0.885	0.868	–	–
Avg(*, 90)	1.081	0.910	0.863	0.866	0.857	0.855	0.858	0.855	0.853	–
Avg(*, 100)	1.076	0.896	0.865	0.860	0.823	0.849	0.844	0.846	0.835	0.836

TS-Swap

Avg(0.6, *)	1.315	1.072	1.009	0.982	0.977	0.977	0.983	0.983	0.983	0.979
Avg(0.8, *)	1.392	1.136	1.089	1.064	1.105	1.123	1.129	1.126	1.138	1.139
Avg(1.0, *)	1.404	1.129	1.092	1.072	1.095	1.096	1.107	1.111	1.114	1.112
Avg(1.2, *)	1.413	1.153	1.091	1.083	1.095	1.101	1.102	1.105	1.105	1.104
Avg(1.4, *)	1.460	1.167	1.093	1.077	1.059	1.073	1.074	1.068	1.074	1.073
Avg(*, *)	1.397	1.131	1.073	1.060	1.065	1.074	1.080	1.079	1.083	1.081
Avg(*, 10)	1.016	–	–	–	–	–	–	–	–	–
Avg(*, 20)	1.052	1.006	–	–	–	–	–	–	–	–
Avg(*, 30)	1.056	1.025	1.014	–	–	–	–	–	–	–
Avg(*, 40)	1.256	1.051	1.040	1.050	–	–	–	–	–	–
Avg(*, 50)	1.310	1.041	1.015	1.036	1.023	–	–	–	–	–
Avg(*, 60)	1.363	1.099	1.041	1.039	1.042	1.053	–	–	–	–
Avg(*, 70)	1.453	1.139	1.069	1.033	1.055	1.074	1.046	–	–	–
Avg(*, 80)	1.453	1.140	1.062	1.039	1.051	1.069	1.085	1.071	–	–
Avg(*, 90)	1.516	1.173	1.062	1.030	1.039	1.047	1.077	1.073	1.058	–
Avg(*, 100)	1.484	1.189	1.070	1.050	1.047	1.056	1.068	1.078	1.089	1.074

모든 R 값과 모든 작업수 N에 대한 비교값의 평균을 의미한다.

먼저 이웃해 생성 방법에 따른 해의 결과를 살펴보면 다음과 같다. TS-Insert의 결과가 TS-Swap의 결과에 비해 훨씬 좋음을 알 수 있는데, 그 첫번째 이유는 4.2.1과 4.2.2절에서 언급한 바와 같이 삽입이동에 의한 이웃해 생성수가 교환이동에 의한 경우보다 많으므로 현재해로부터 좋은 해로 이동할 확률이 크기 때문이다. 두번째 이유는 교환이동의 경우 현재해로부터 생성되는 이웃해들은 서로 다른 두 작업의 위치를 교환함으로써 이루어지는데, 이 두 작업 중 작업순서상 뒤쪽에 위치해 있는 작업의 납기지연은 줄어들지만 앞쪽에 위치해 있는 작업은 납기지연이 크게 발생할 가능성이 있으므로 이웃해의 질이 서로 상쇄되어 좋은 해로 이동하려는 성질을 방해하기 때문이다.

TS-Hybrid는 삽입이동과 교환이동을 번갈아 가면서 수행하기 때문에 교환이동의 영향으로 TS-Insert에 비해 해의 향상속도가 늦지만, 탐색 영역이 늘어날수록 TS-Insert보다 더 좋은 해를 갖게 된다. 한 가지의 이웃해 생성 방법만 사용한다면 탐색이 진행됨에 따라 어느 정도의 해의 이동이 이루어 진 이후에도 해가 더 좋아지지 않을 경우 해의 순환이 발생할 확률이 커지게 되지만, TS-Hybrid는 삽입이동과 혼합이동을 교대로 사용하기 때문에 해의 순환을 막아주고 다양하게 해를 찾아감으로 인하여 한 가지 방법에 의해서 이웃해를 생성해내는 것보다 더 좋은 결과를 보여준다.

R 값에 따른 해의 결과를 살펴보면 전체적으로 R 값이 작을수록 본 연구에서 제시한 알고리듬의 해가 좋다는 것을 알 수 있다. 이것은 RHP 알고리듬이 R 값과 관련하여 갖는 취약점으로 R 값이 작을 경우, 즉 스케줄 시점에 기계 앞에서 대기하고 있는 가용한 작업수가 많을 경우 그렇지 않은 경우 보다 매우 좋지 않은 결과를 보인다는 것이다. 그 이유는 RHP 알고리듬의 절차 중 예측구간 내의 작업들을 대상으로 납기순으로 최대 k개의 작업을 선정하고, 선정된 작업들을 대상으로 분지한계법을 이용하여 최적해를 구해 부분 스케줄을 확정짓는데, 여기서 분지한계법의 수행속도의 한계로 인해 k 값이 10 이하로 설정되기 때문이다. 따라서, 스케줄 시점에 가용한 작업수가 10을 초과할 때 이 작업들 중 납기순으로 최대 10개만을 선정하여 최적해를 구하는 것은 전체 해의 질을 떨어뜨리는 요인이 될 수 있다. 그 다음으로 작업수 N에 따른 해의 결과를 보면 작업수가 커질수록 TS 알고리듬의 해가 우수하다는 것을 알 수 있는데, 이것 역시 RHP 알고리듬이 R 값과 관련하여 갖는 취약점으로 작업수 N이 커질 경우 스케줄 시점에 기계 앞에서 대기하고 있는 가용한 작업수가 상대적으로 많아지는 것과 관련이 있다.

탐색깊이와 해의 질의 관계는 <그림 7>에 나타나 있는데, 탐색깊이 값이 커질수록 탐색강도가 커지므로 인해 해의 향상이 존재한다는 것을 보여주고 있다. 그러나 탐색깊이 값이 일정한 값에 도달하게 되면 해의 향상 정도가 거의 없는 반면에 <그림 8>에서와 같이 알고리듬 수행 속도는 탐색깊이 값에 따라 일정하게 비례하여 증가한다는 점에서 효율적인 탐색깊이

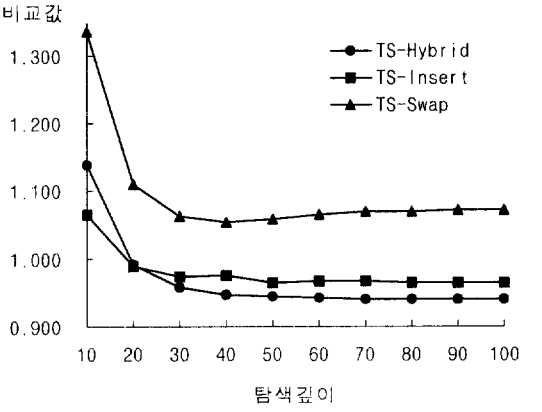

그림 7. 탐색깊이에 따른 해의 결과.

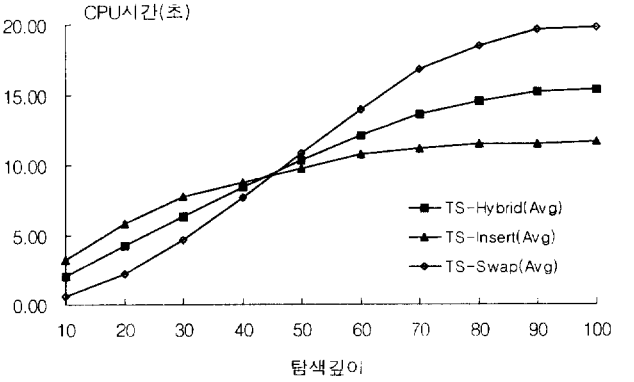

그림 8. 탐색깊이에 따른 평균 수행속도.

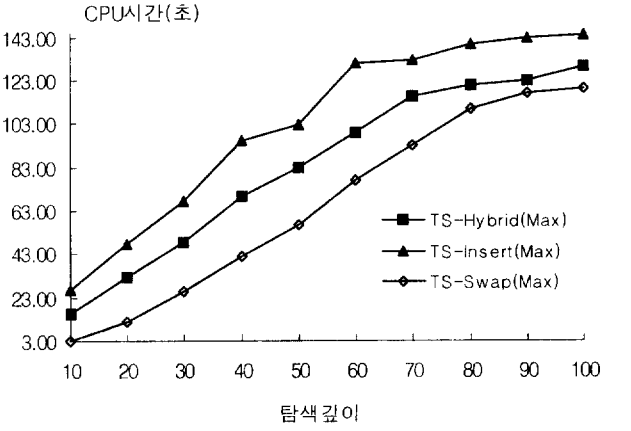

그림 9. 탐색깊이에 따른 최대 수행속도.

를 결정하는 것이 필요하다는 것을 실험결과 알 수 있다. 또한 <그림 8>과 <그림 9>에서 TS-Hybrid(Avg)와 TS-Hybrid(Max)는 TS-Hybrid 알고리듬을 사용하였을 때 사용된 평균 CPU 시간과 최대 CPU 시간을 각각 의미한다.

<그림 10>은 작업수가 100인 문제집합에 대해 TS 알고리듬으로 얻은 결과가 O&U해에 도달하는 데 까지 소요된 탐색 반복횟수와 CPU 시간을 보여주고 있다. 본 실험에서는 RHP 알고리듬과 TS 알고리듬의 수행속도를 비교하기 위해 동일한

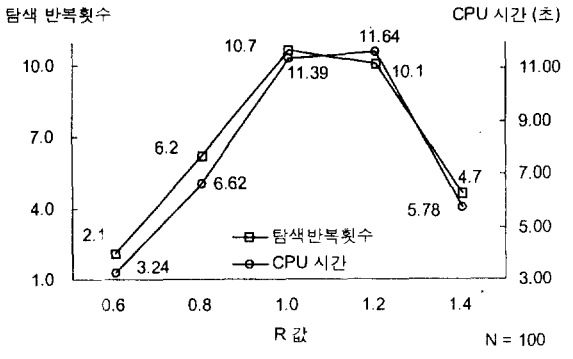

그림 10. O&U해까지 소요된 탐색 반복횟수 및 시간.

컴퓨터에서 분지한계법을 사용하여 작업수 10인 문제를 CPLEX 상에서 실험하였는데, 그 결과 작업수 10인 문제를 푸는 데 소요된 시간은 평균 2.12초였다. 따라서, RHP 알고리듬은 작업수 100인 문제를 풀기 위해 λ 값($\lambda = 1, 2, 3$)에 따라 작업수 10인 문제를 최소 24번에서 최대 91번 풀어야한다는 것을 감안할 때, 평균 7.70초의 시간이 소요되는 TS 알고리듬은 RHP 알고리듬에 비해 약 7배에서 25배 이상의 매우 빠른 수행속도를 나타낸다는 것을 실험결과 알 수 있다.

<표 4>는 마지막으로 전체적인 TS 알고리듬의 성능을 확인하기 위하여 1000개의 문제집합에 대해 TS 알고리듬에 의해 얻은 가장 좋은 해(TS 알고리듬(Best))를 기준으로 RHP 알고리듬의 O&U해와 비교하여 우수한 결과를 보인 횟수, 동일한 결과를 보인 횟수, 좋지 않은 결과를 보인 횟수를 비율로 나타낸 것이다.

지금까지 분석한 결과들을 종합해보면 본 연구에서 제시한 TS 알고리듬은 해의 질과 수행속도 면에서 RHP 알고리듬보다 매우 우수하며, 특히 기계 앞에서 대기하고 있는 가용한 작업수가 많을수록(R 값이 작을수록), 작업수가 큰 문제일수록 TS 알고리듬은 보다 탁월한 성능을 보여준다는 것을 알 수 있다.

표 4. TS 알고리듬의 최고(Best) 성능

	TS 알고리듬 (Best)	우수한 경우	동일한 경우	열등한 경우
Avg(0.6, *)	0.816	81.5%	17.0%	1.5%
Avg(0.8, *)	0.887	68.5%	30.5%	1.0%
Avg(1.0, *)	0.940	44.5%	54.5%	1.0%
Avg(1.2, *)	0.983	18.0%	80.5%	1.5%
Avg(1.4, *)	0.994	7.5%	92.5%	0.0%
Avg(*, 10)	1.007	0.0%	96.0%	4.0%
Avg(*, 20)	0.983	17.0%	82.0%	1.0%
Avg(*, 30)	0.936	34.0%	66.0%	0.0%
Avg(*, 40)	0.909	44.0%	55.0%	1.0%
Avg(*, 50)	0.876	49.0%	50.0%	1.0%
Avg(*, 60)	0.862	58.0%	41.0%	1.0%
Avg(*, 70)	0.831	59.0%	40.0%	1.0%
Avg(*, 80)	0.831	57.0%	43.0%	0.0%
Avg(*, 90)	0.801	61.0%	38.0%	1.0%
Avg(*, 100)	0.779	61.0%	39.0%	0.0%
Avg(*, *)	0.882	44.0%	55.0%	1.0%

6. 결론 및 추후 연구

본 연구에서는 순서 의존적인 작업 준비시간과 작업 투입시점이 존재하는 동적인 환경에서 L_{max}를 최소화하는 단일기계 일정계획 수립 방법을 제시하였다. 본 논문에서 제시된 방법은 초기해를 구하는 방법과 초기해를 개선하는 알고리듬으로 구성되어 있으며, 초기해를 개선하는 알고리듬은 메타 휴리스틱 중의 하나인 타부 탐색을 이용하였다. 제시된 알고리듬의 객관적인 성능 평가를 위하여 벤치마킹 데이터(Uzsoy)를 사용하여 기존의 RHP 알고리듬에 비해 빠른 시간 내에 좋은 해를 찾음을 보여주었고, 특히 작업수가 증가할수록 작업 투입시점이 집중되어 있는 환경일수록 매우 우수한 해를 제공한다는 것을 입증하였다. TS 알고리듬은 결정모수인 탐색 반복횟수와 탐색깊이의 값을 변화시켜 사용함으로써 해의 값을 해의 질과 수행속도 면에서 효율적으로 유연하게(Flexible) 조절하여 구할 수 있을 뿐만 아니라, 작업의 우선 순위와 같은 현실적인 조건이 향후 더 추가된다 하더라도 쉽게 수정하여 사용될 수 있다.

추후 연구로는 TS 알고리듬을 단일 기계에서 뿐만 아니라 더 복잡한 문제인 병렬 기계 또는 잡샵 환경으로 확장하여 적용하는 연구가 요구되어진다.

참고문헌

Ali, A., Jatinder, N. D. G. and Tariq, A.(1999), A review of scheduling research involving setup considerations. *The International Journal of Management Science*, **27**, 219-239.

Baker, K. R. and Su, Z. S. (1974), Scheduling with due dates and early start times to minimize maximum tardiness, *Naval Research Logistics Quarterly*, **21**, 171-176.

Bianco, L., Ricciardelli, S., Rinaldi, G. and Sassano, A. (1988), Scheduling tasks with sequence-dependent processing times, *Naval Research Logistics Quarterly*, **35**, 177-184.

Carlier, J. (1982), The one-machine scheduling problem, *European Journal of Operational Research*, **11**, 42-47.

Crauwels, H. A. J., Potts, C. N. and Wassenhove, L. N. V. (1996), Local search heuristics for the single machine scheduling with batching to minimize the number of late jobs, *European Journal of Operational Research*, **90**, 200-213.

Farn, C. D. and Muhlemann, A. P. (1979), The dynamic aspects of a production scheduling problem, *International Journal of Production Research*, **17**, 15-21.

Garey, M. R. and Johnson, D. S. (1979), *Computers and intractability: A guide to the theory of NP completeness*, W. H. Freeman and Company, San Francisco.

Glover, F. (1989), Tabu search-Part I, *ORSA Journal on Computing*, **1**, 190-206.

Glover, F. (1990), Tabu search-Part II, *ORSA Journal on Computing*, **2**, 4-32.

Kim, S. Y., Lee, Y. H. and Agnihotri, D. (1995), A hybrid approach to sequencing jobs using heuristic rules and neural networks, *Production Planning Control*, **6**, 445-454.

Laguna, M., Barnes, J. W. and Glover, F. (1991), Tabu search methods for single machine scheduling problem, *Journal of Intelligent Manufacturing*, **2**, 63-74.

Laguna, M. and Gonzalez, V. J. L. (1991), A search heuristic for just-in-time scheduling in parallel machines, *Journal of Intelligent Manufacturing*, **2**, 253-260.

Lee, Y. H., Bhaskran, K. and Pinedo, M. (1997), A heuristic to minimize the total weighted tardiness with sequence-dependent setups, *IIE Transactions*, **29**, 45-52.

Ovacik, I. M. and Uzsoy, R. (1994a), Exploiting shop floor status information to schedule complex job shops, *Journal of Manufacturing Systems*, **13**, 369-388.

Ovacik, I. M. and Uzsoy, R. (1994b), Rolling Horizon Algorithms for a Single-Machine Dynamic Scheduling Problem with Sequence Dependent Setup Times, *International Journal of Production Research*, **32**, 1243-1263.

Ovacik, I. M. and Uzsoy, R. (1997), *Decomposition methods for complex factory scheduling problems*, Kluwer Academic Publishers, Boston/Dordrecht/London.

Uzsoy, R., Lee, C. Y. and Martin-Vega, L. A. (1992), Scheduling semiconductor test operations: minimizing maximum lateness and number of tardy jobs on a single machine, *Naval Research Logistics*, **39**, 369-388.

Uzsoy, R., *http://palette.ecn.purdue.edu/~uzsoy2/Problems/single/parameters.html*.