韓國經營科學會誌
第21卷 第2號
1996年 8月

페트리네트를 이용한 FMS스케줄링에 대한 발견적 해법

안재홍* · 노인규**

A Heuristic Algorithm for FMS Scheduling Using the Petri Net

Jae-Hong Ahn* · In-Kyu Ro**

Abstract

The main purpose of this study is to develop an algorithm to solve the scheduling problems of FMS using Petri-net which describes exactly features of discrete event system. Petri-net is well suited to model the dynamics of FMS and Petri-net is an ideal tool to formulate scheduling problems with routing flexibility and shared resources. By using the marking of Petri-net, We can model features of discrete event system, such as concurrency, asynchronous, conflict and non-determinism. The proposed algorithm in this paper can handle back-tracking using the marking of Petri-net. The results of the experiment show that marking is one of the best ways that describe exactly movement of the discrete event system. To show the effectiveness of the algorithm suggested here, we compare it with L1 algorithm using the Petri-net through the test on randomly generated test problems.

1. 서 론

오늘날 제조업은 여러가지 어려움에 직면하고 있다. 이러한 어려움은 고객의 요구가 다양해지고, 제품의 수명주기가 짧아지며, 제조기술이 급변하는 것으로 부터 기인한다. 이러한 어려움을 해결하기 위해 제조업계에서는 생산시스템의 효율성과 유연성의 향상이 중요시되고 있으며, 효율성이 뛰어난 흐름생산방식 (Flow shop) 과 유연성이 뛰어난 주문생산방식 (Job shop) 의 장점을 모두 갖추고 있는 유연생산시스템 (Flexible Manufacturing System, FMS) 에 관심이 모아지고 있다.

* 한양대학교 산업공학과
** 한양대학교 산업공학과

이러한 FMS 를 제어하고 스케줄링하기 위해서는 정수계획법같은 수리적방법보다 시스템의 동적인 변화를 민감하게 표현할 수 있는 방법의 필요성이 높아지고 있다. 이러한 요구에 부응할 수 있는 FMS 스케줄링 방법을 제시하고자 한다.

본 연구는 두가지의 목적을 가지고 있다. 첫번째 목적은 페트리네트를 작성하는데 요구되어지는 도달가능 그래프 (reachability graph)를 부분적으로 작성하는 것이고, 두번째 목적은 총작업수행시간을 최소화하기 위해 최적자원배분 (optimal resource allocation)에 대한 효율적인 스케줄링 방법을 개발하는 것이다.

페트리네트를 이용하여 스케줄링 하는 방법은 유연성이 높은 가공 공정을 다루기 쉽고, 스케줄링 문제에 나타나는 이산사건들을 정확하게 표현하는 것이 가능하며, 페트리네트의 특징인 시스템정지 (dead-lock) 에 대한 예방을 보장하므로 분석이 용이하다. 또한 페트리네트의 마킹을 사용하여 back tracking을 한다는 점에 의의가 있다.

페트리네트는 1962년에 Carl A. Petri [1]에 의해 처음으로 통신시스템을 모델링하고 해석하기 위해서 개발되었다. 이 연구는 비동기적 (asynchronous) 통신흐름정보 및 비동기적인 시스템의 모델링에 기반이 되었으며, 그 후 1960년대 말에 MIT에서 페트리네트의 조건 및 비동기성에 대한 연구를 수행하였으며, 모델의 성능을 평가하기 위하여 페트리네트에 시간의 개념을 도입하기 시작하였다.

Commoner et al. [18] 는 페트리네트를 이용하여 시스템정지 (deadlock) 에 대해 연구하였으며, 최근에는 FMS 를 모델링하고 평가하는 도구로 페트리네트를 널리 이용하고 있다 [4][6][8][9][14][17].

시간사양 페트리네트 (Timed Petri-net)는 1974년에 개발되었고, Dubois 와 Stecke [10]는 시간사양 페트리네트를 사용해서 시스템의 가동률, 기계의 사용률 등과 같은 항목을 평가하였으며, Jothi-shankar 와 Wang [7] 은 시간사양 페트리네트를 사용하여 최적 간판수를 결정하는 문제를 해결하였다. Ciardo 와 Lindemann [12]. Boucherie [13], Lin 과 Marinescu [15]는 시간사양 페트리네트의 발전된 개념인 확률적인 페트리네트 (Stochastic Petri-net) 를 FMS 모델링에 적용하였다.

1981년에 Jensen [11]이 색사양 페트리네트 (Colored Petri-net)를 개발하였고, Huang 과 Chang [6], Cossins 과 Ferreira [16]는 Colored 페트리네트를 사용해서 FMS 를 제어하는 방법을 개발하였다.

1992년에 Dicesare 와 Lee [3]는 페트리네트를 FMS 스케줄링에 적용하여 L1 algorithm을 개발하였고, Rodammer 와 White [5]는 스케줄링에 관한 참고논문에서 페트리네트를 언급하였으며, Matsuura et al. [2]는 시뮬레이션이나 페트리네트 같은 모델링기법을 스케줄링 문제에 이용하는 것이 보다 효율적임을 제안하였다

2. 페트리네트

페트리네트는 여러 특성을 가진 시스템을 모델링할 수 있는 기법이다. 페트리네트는 그래픽 기능과 수학적 해석능력을 가지고 있으며 토큰의 흐름을 통하여 시스템내에 존재하는 개체의 흐름을 그림으로 파악할 수 있고, Markov chain 을 사용해서 기계의 효율, 평균

대기열의 길이, 평균대기시간, 총작업수행시간과 같은 수행도평가기준(performance measure)을 구할 수 있다 [19][20][21].

시스템을 모델링하는 다른 방법으로는 Queueing networks 와 시뮬레이션이 있는데, Queueing networks 는 세부적인 사항을 정확하게 모델링하기가 쉽지 않지만 일단 모델링이 되면 수학적인 해석이 명확하고, 시뮬레이션은 비교적 모델링 하기가 쉽지만 정확한 해석을 하려면 많은 수행시간을 필요로 한다.

페트리네트는 시스템의 변화를 그래픽으로 나타내기 때문에 이해하기 쉽고, 소프트웨어로 개발하는 것이 가능하며, 시뮬레이션에 비해 많은 계산시간을 요구하지 않고, 이산사건시스템을 모델링할 수 있으며, 최적 버퍼크기를 결정하는 경우나 병목공정을 찾는 경우, 또는 FMS 스케줄링 문제에 매우 유용하게 사용할 수 있다.

3. 알고리즘 및 예제

이 장에서는 페트리네트의 마킹을 사용하여 back tracking 을 하는 스케줄링 알고리즘을 제시하고자 한다. 3. 1절에서는 알고리즘에 사용되는 기호를 정의하였고, 3. 2절에서는 알고리즘을 제시하였다. 또한 3. 3절과 3. 4절에서는 예를 들어 자세히 설명하였다.

3. 1 기호정의

t : 현재 시각

p_i : 플레이스 i

t_j : 트랜지션 j

$I(p_i, t_j)$=0, p_i 에서 t_j 로 연결된 아크가 없는 경우

1, p_i 에서 t_j 로 연결된 아크가 있는 경우

I(P, T) : 입력함수

O(P, T) : 출력함수

P : 플레이스의 전체 집합

T : 트랜지션의 전체 집합

$O(p_i, t_j)$=0, t_j 에서 p_i 로 연결된 아크가 없는 경우

1, t_j 에서 p_i 로 연결된 아크가 있는 경우

M(0) : 초기마킹

M(k) : k번째 생성된 마킹

M(f) : 종료마킹

M' : 활성화된 마킹중에서 가장 작은 가공시간을 가지는 마킹

m_a : 기계 a (a는 기계의 번호)

Set_a [ ] : 활성화규칙에 의해서 기계 a가 선택된 경우의 마킹을 기록하기 위한 배열

T [ ] : 방출시간을 기록하기 위한 배열

3. 2 스케줄링 알고리즘

FMS 를 스케줄링하는 경우에 사용할 수 있는 기계가 없는 경우에 작업을 지연시키는 것이 효과적이라고 볼 수 없다. 이러한 경우에 기계를 사용할 수 있도록 이전의 단계로 거슬러 올라가서 스케줄을 변경시켜 작업의 지연을 막고자한다. 본 알고리즘은 총작업수행시간을 수행도평가기준으로 하였으며, 알고리즘의 구체적인 절차는 다음과 같다.

단계1. 초기마킹 M(0)를 현재마킹으로 선택한다.

단계2. 현재마킹의 하부마킹 중에 활성화된 마킹이 있는지 검색한다.

단계3. 3-1. 활성화된 마킹이 있으면 활성화된 마킹 중에서 가장 작은 가공시간을 가지는 하부마킹을 M'로 선택한다.

3-2. 활성화된 마킹이 없으면 (활성화되지 않은) 모든 하부마킹 에서 최소의 가공시간을 가지는 트랜지션이 사용하는 기계 m_i를 택하여, Seti의 마지막 마킹을 M'로 선택한다.

단계4. 현재마킹에 대한 활성화시간과 방출시간을 구한다.

단계5. 방출시간을 배열 T[ ]에 저장하고 배열 T[ ]를 오름차순으로 정렬한다.

단계6. 사용하는 기계가 m_i 이면 현재마킹을 Set_i 에 넣는다.

단계7. M'에 대한 활성화된 마킹이 더 존재하는지 검색하여, 존재하면 단계3의 3-1을 수행한다.

단계8. 시간 t가 배열 T[ ]의 첫번째 원소와 같으면, 방출된 마킹과 종료마킹 M(f)를 비교한다. 방출된 마킹과 종료마킹이 같으면 알고리즘을 종료하고, 다르면 단계2를 수행한다.

3. 3 예제

3. 2절에서 제시한 알고리즘에 대한 하나의 예제를 만들어 총작업수행시간과 트랜지션의 방출순서를 구해보자. 예제는 다음과 같다.

m1, m2, m3의 3대의 기계를 가지는 FMS를 스케줄링하려고 한다. 이 시스템은 J_1, J_2의 두개의 작업을 다룬다. J_1은 O_{11}, O_{12}, O_{13}의 세개의 공정으로 이루어져 있으며, J_2은 O_{21}, O_{22}, O_{23}의 세개의 공정으로 이루어져 있다.

표3. 1은 각각의 공정이 어느 기계에 의해 가공되어질 수 있는지 나타내고 있으며, 표3. 2는 각각의 공정에 대한 기계의 가공시간을 나타낸다.

작업1의 첫번째 공정 (O_{11}) 에 사용될 수 있는 기계는 m1, m2 두대임을 표3. 1에서 알 수 있고, m1을 사용할 경우의 가공시간은 3이고, m2를 사용할 경우의 가공시간은 4임을 표 3. 2를 통해서 알 수 있다.

〈표3. 1〉 각각의 공정과 사용가능한 기계

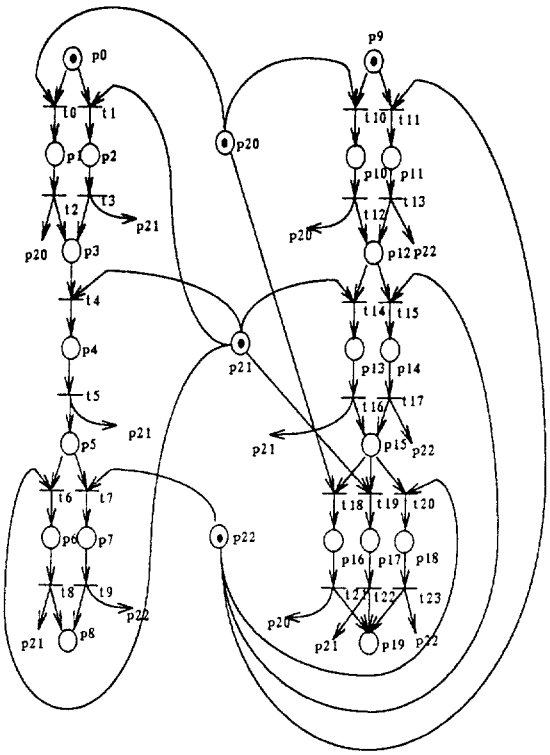
작업	첫번째 공정(O_{i1})	두번째 공정(O_{i2})	세번째 공정(O_{i3})
J1	m1, m2	m2	m2, m3
J2	m1, m3	m2, m3	m1, m2, m3

표3. 2 트랜지션의 가공시간

공정(사용기계)	가공시간
O_{11}(m1)	3
O_{11}(m2)	4
O_{12}(m2)	3
O_{13}(m2)	3
O_{13}(m3)	2
O_{21}(m1)	4
O_{21}(m3)	2
O_{22}(m2)	4
O_{22}(m3)	5
O_{23}(m1)	3
O_{23}(m2)	4
O_{23}(m3)	4

3. 4 예제의 페트리네트 모델

이 시스템에 대한 페트리네트 모델은 그림3. 1에 나타나 있고, 페트리네트에 사용된 각각의 플레이스에 대한 설명은 표3. 3에 나타나 있다.



〈그림 3. 1〉 예제 FMS에 대한 페트리네트 모델

3. 3 플레이스에 대한 설명

플레이스	설 명
P_0	작업1 준비
P_1	O_{11} 을 m_1에서 가공
P_2	O_{11} 을 m_2에서 가공
P_3	공정 O_{11} 완료
P_4	O_{12} 을 m_2에서 가공
P_5	공정 O_{12} 완료
P_6	O_{13} 을 m_2에서 가공
P_7	O_{13} 을 m_3에서 가공
P_8	작업1 완료
P_9	작업2 준비
P_{10}	O_{21} 을 m_1에서 가공
P_{11}	O_{21} 을 m_3에서 가공
P_{12}	공정 O_{21} 완료
P_{13}	O_{22} 을 m_2에서 가공
P_{14}	O_{22} 을 m_3에서 가공
P_{15}	공정 O_{22} 완료
P_{16}	O_{23} 을 m_1에서 가공
P_{17}	O_{23} 을 m_2에서 가공
P_{18}	O_{23} 을 m_3에서 가공
P_{19}	작업2 완료
P_{20}	기계1 이용가능
P_{21}	기계2 이용가능
P_{22}	기계3 이용가능

예제에 대한 입력함수는 그림3. 2와 같다. 각 행은 플레이스를 나타내며, 각 열은 트랜지션을 나타낸다. 첫번째 열은 트랜지션 t_0를 의미하며, 첫번째 열의 1번째 값과 21번째 값이 1의 값을 가지는데 이것은 플레이스 p0와 플레이스 p_{20}이 트랜지션 t_0의 입력플레이스임을 의미한다.

```
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0
0 1 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0
```

그림3. 2 예제에 대한 입력함수

그림3. 3는 예제에 대한 출력함수를 나타내는데, 입력함수와 마찬가지로 각 행은 플레이스를 의미하며, 각 열은 트랜지션을 의미한다. 첫번째 열의 2번째 값만 1의 값을 가지는데 이것은 트랜지션 t_0가 토큰을 방출하였을 경우에 플레이스 p1에 토큰이 입력되어짐을 의미한다.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0
0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1
```

그림3. 3 예제에 대한 출력함수

다음 절에서는 3. 3절에서 제시한 예제를 각 단계별로 수행하여 알고리즘을 보다 자세히 살펴보자.

3. 5 예제의 수행

예제를 단계별로 수행하면 다음과 같다. 총 9개의 iteration으로 이루어져 있으며, 총작업수행시간은 10이다.

iteration 1.

단계0. t=0

단계1. M(0) 10000000010000000000111 를 현재 마킹으로 선택한다.

단계2. 활성화된 마킹이 있는지 검색한다.

M(1)01000000010000000000011 3

M(2)00100000010000000000101 4

M(3)10000000001000000000011 4

M(4)10000000000100000000110 2

단계3. 가장 작은 가공시간을 갖는 M(4)를 M'로 선택한다.

단계4. 활성화 시간 : t=0

방출 시간 : 2

단계5. 방출 시간을 배열 T에 넣는다.

T[ ]={2}

단계6. m_3을 사용하므로 set3 에 현재마킹을 넣는다.

set1 { }

set2 { }

set3 {M(0)10000000010000000000111}

단계7. M(4)를 현재마킹으로 선택하여, 활 성화된 마킹이 있는지 검색한다.

M(5)01000000000100000000010 3

M(6)00100000000100000000100 4

단계3. 가장 작은 가공시간을 갖는 M(5)를 M'로 선택한다.

단계4. 활성화 시간 : t=0

방출 시간 : 3

단계5. 방출 시간을 배열 T에 넣는다.

T[ ]={2, 3}

단계6. m_1을 사용하므로 set_1 에 현재마킹을 넣는다.

set1 {M(4)10000000000100000000110}

set2 { }

set3 {M(0)10000000010000000000111}

iteration 2.

단계0. t=2

단계1. M(7)01000000000010000000011 를 현재 마킹으로 선택한다.

단계2. 활성화된 마킹이 있는지 검색한다.

M(8)01000000000001000000001 4

M(9)01000000000000100000010 5

단계3. 가장 작은 가공시간을 갖는 M(8)를 M'로 선택한다.

단계4. 활성화 시간 : t=2

방출 시간 : 4

단계5. 방출 시간을 배열 T에 넣는다.

T[ ]={3, 6}

단계6. m_2를 사용하므로 set2 에 현재마킹을 넣는다.

set1 {M(4) 10000000000100000000110}

set2 {M(7) 01000000000010000000011}

set3 {M(0) 10000000010000000000111}

iteration 3.

단계0. t=3

단계1. M(10)00010000000001000000101 를 현재 마킹으로 선택한다.

단계2. 활성화된 마킹이 있는지 검색한다.

단계3. 활성화된 마킹이 없지만 m_2가 사용가능하다면 작업을 지연시키지 않아도 된다. set_2의 마지막 마킹을 현재마킹으로 선택하여 다시 계산한다. 단 m_2는 선택할 수 없다.

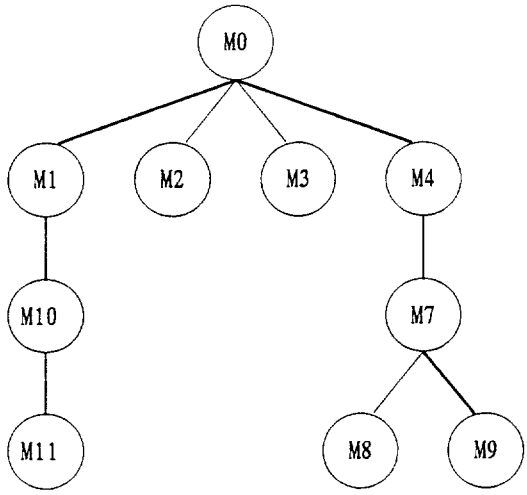
iteration 4.

단계0. t=2

단계1. M(7)01000000000010000000011 를 현재 마킹으로 선택한다.

단계2. 활성화된 마킹이 있는지 검색한다.
M(8)0100000000000010000000001 4
M(9)0100000000000000100000010 5
단계3. M(8)를 제외한 가장 작은 가공시간을 갖는 M(9)를 M'로 선택한다.
단계4. 활성화 시간 : t=2
방출 시간 : 5
단계5. 방출 시간을 배열 T에 넣는다.
T[ ]={3, 7}
단계6. m_3을 사용하므로 set_3 에 현재마킹을 넣는다.
set1 {M(4)10000000000100000000110}
set2 { }
set3 {M(0)10000000010000000000111,
M(7)01000000000010000000011}

iteration 5.
단계0. t=3
단계1. M(10)00010000000000100000110 를 현재 마킹으로 선택한다.
단계2. 활성화된 마킹이 있는지 검색한다.
M(11)0000100000000100000100 4
단계3. 가장 작은 가공시간을 갖는 M(11)을 M'로 선택한다.
단계4. 활성화 시간 : t=3
방출 시간 : 3
단계5. 방출 시간을 배열 T에 넣는다.
T[ ]={6, 7}
단계6. m_2를 사용하므로 set_2 에 현재마킹을 넣는다.
set1 {M(4)10000000000100000000110}
set2 {M(10)00010000000000100000110}
set3 {M(0)10000000010000000000111,
M(7)01000000000010000000011}



〈그림 3. 4〉 iteration 5에서의 도달가능그래프

iteration 6.
단계0. t=6
단계1. M(12)00000100000000100000110 를 현재 마킹으로 선택한다.
단계2. 활성화된 마킹이 있는지 검색한다.
M(13)0000001000000010000100 3
단계3. 가장 작은 가공시간을 갖는 M(13)을 M'로 선택한다.
단계4. 활성화 시간 : t=6
방출 시간 : 3
단계5. 방출 시간을 배열 T에 넣는다.
T[ ]={7, 9}
단계6. m_2를 사용하므로 set_2 에 현재마킹을 넣는다.
set1 {M(4)10000000000100000000110}
set2 {M(10)00010000000000100000110,
M(13)0000001000000010000100}
set3 {M(0)10000000010000000000111,
M(7)01000000000010000000011}

iteration 7.

단계0. t=7

단계1. M(14)00000010000000001000101 를 현재 마킹으로 선택한다.

단계2. 활성화된 마킹이 있는지 검색한다.
M(15)000000100000000000100001 3
M(16)00000010000000000001100 4

단계3. 가장 작은 가공시간을 갖는 M(15)을 M'로 선택한다.

단계4. 활성화 시간 : t=7
방출 시간 : 3

단계5. 방출 시간을 배열 T에 넣는다.
T[ ]={9, 10}

단계6. m_1을 사용하므로 set_1 에 현재마킹을 넣는다.
set1 {M(4)10000000000100000000110,
M(16)00000010000000000001100}
set2 {M(10)00010000000000100000110,
M(13) 00000010000000100000100}
set3 {M(0)10000000010000000000111,
M(7)01000000000010000000011}

iteration 8.

단계0. t=9, 작업1의 마지막 공정이 끝나서 더 이상 작업 1는 고려하지 않아도 된다.

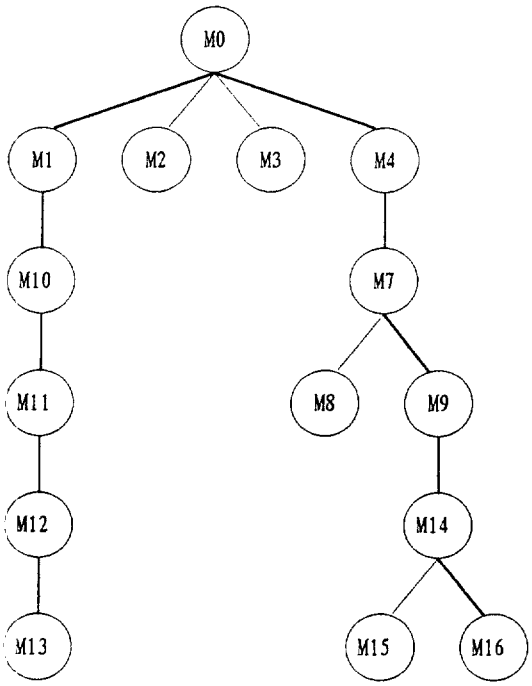
단계1. M(17) 00000000100000010000011이 현재의 마킹이다.

iteration 9.

단계0. t=10, 작업2의 마지막 공정이 끝나서 더이상 작업2는 고려하지 않아도 된다.

단계1. M(18)00000000100000000001111이 현재의 마킹이다. 여기서 M(18)이 종료마킹과 동일하므로 알고리즘을 종료한다.

Iteration 9에서 알고리즘이 종료되며, iteration 9까지 수행한 도달가능 그래프를 그리면 그림3. 5과 같다.



〈그림 3. 5〉 예제에 대한 도달가능 그래프

3. 6절에는 알고리즘의 수행결과에 대한 보다 구체적인 결과가 제시되어 있다.

3. 6 예제 결과

$A_1(M)$은 마킹을 사용하여 back tracking 을 하는 알고리즘이고, $A_2(M)$은 back tracking 을 하지 않는 알고리즘이다. 본 예제를 두가지 방법으로 수행해본 결과는 표3. 4, 표3. 5에 나타나 있다.

표3. 4에서 각 알고리즘에 대한 최적스케줄과 총작업수행시간을 알 수 있는데, $A_1(M)$의

총작업수행시간은 10이고 $A_2(M)$의 총작업수행시간은 11이다. 그리고 표3. 5에서는 각 알고리즘의 iteration수와 트랜지션의 방출순서를 알 수 있다.

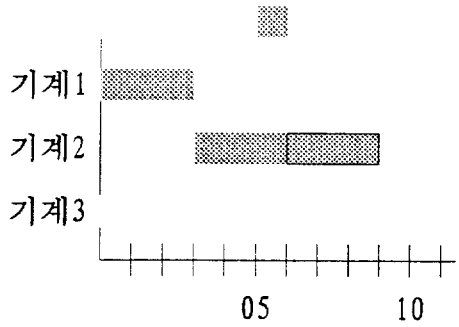
〈표 3. 4〉 예제에 대한 스케줄

$A_1(M)$		$A_2(M)$	
최적순서	가공시간	최적순서	가공시간
O_{21} m_3	2	O_{21} m_3	2
O_{11} m_1	3	O_{11} m_1	3
O_{22} m_3	5	O_{22} m_2	4
O_{12} m_2	3	O_{12} m_2	3+3
O_{13} m_2	3	O_{13} m_3	2
O_{23} m_1	3	O_{23} m_1	3
총작업 수행시간	10	총작업 수행시간	11

〈표 3. 5〉 예제의 수행결과

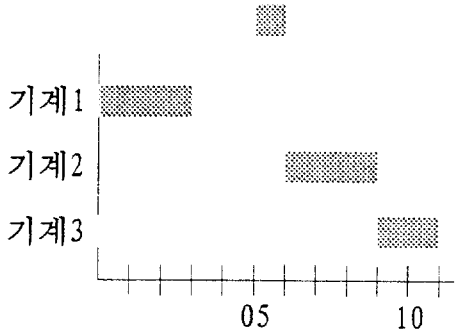
	iteration 수	총작업 수행시간	트랜지션 방출순서
$A_1(M)$	9	10	t_{11} t_0 t_{13} t_{15} t_2 t_4 t_5 t_7 t_{17} t_{18} t_9 t_{21}
$A_2(M)$	7	11	t_{11} t_0 t_{13} t_{15} t_2 t_4 t_{17} t_{20} t_5 t_6 t_{23} t_8

그림3, 6는 예제결과에 대한 $A_1(M)$의 간트차트를 나타낸다. x축은 시간을 나타내며, y축은 사용되는 기계를 나타낸다. 기계1의 이용율은 60%, 기계2의 이용율은 60%, 기계3의 이용율은 70% 임을 알 수 있고, 작업1의 첫번째공정 (O_{11})은 기계1에 의해서 가공되었고, 두번째 공정 (O_{12})은 기계2에 의해서 가공되었으며, 세번째 공정 (O_{13})은 기계3에 의해 가공되었음을 알 수 있다.



〈그림 3. 6〉 예제결과에 대한 $A_1(M)$의 간트차트

마찬가지로 그림 3. 7은 예제결과에 대한 $A_2(M)$의 간트차트를 나타내며, 기계1의 이용율은 54%, 기계2의 이용율은 63%, 기계3의 이용율은 36% 임을 알 수 있다.



〈그림 3. 7〉 예제결과에 대한 $A_2(M)$의 간트차트

페트리네트의 마킹을 사용함으로써 시스템의 특정 상태로 되돌아 가는 것이 가능하다. 즉 활성화된 하부마킹중에서 가장 작은 가공시간

을 가지는 트랜지션을 선택하는 방식으로 알고리즘을 계속 수행하는 경우에 작업의 지연이 생길 수도 있는데, 이러한 경우에 페트리네트의 마킹을 사용하는 back tracking을 하여 작업이 지연되지 않도록 스케줄을 변경하고자 한다. 예제의 수행을 통하여 back tracking 을 하는 경우가 back tracking을 하지 않는 경우보다 작업총수행시간이 줄어드는 것을 알 수 있었다. 보다 많은 경우에 대해 4장에서 실험해 보도록 하자.

4. 실 험

본 4절에서는 두가지의 실험을 하고자 한다. 첫번째 실험은 4. 1에서 수행하였으며, back tracking에 대한 필요성에 대해 알아보기 위한 실험이다. 동일한 100개의 문제에 대해 back tracking을 하는 알고리즘과 back tracking을 하지 않는 두개의 알고리즘을 적용하였으며, back tracking을 하였을 경우에 최소총작업수행시간이 줄어들었음을 알 수 있었다. 4. 2에서는 Lee와 Dicesare가 개발한 L1 알고리즘을 소개하였으며, 4. 3에서는 동일한 300개의 문제에 대해 본 연구에서 제시한 알고리즘과 L1 알고리즘을 적용해 보았다.

4. 1 Back tracking이 해에 미치는 영향

앞절의 예제에서 페트리네트의 마킹을 사용하여 시스템의 특정한 상황으로 돌아갈 수 있는 back tracking을 사용하는 경우에 최소총작업수행시간이 줄어듦을 알 수 있었다. $A_1(M)$은 마킹을 사용하여 back tracking 한 알고리즘이고, $A_2(M)$은 back tracking 하지 않은 알고리즘이다.

〈표 4. 1〉 실험 4. 1의 결과

$A_1(M)$의 최소총작업 수행시간	$A_2(M)$의 최소총작업 수행시간	$A_1(M)$의 컴퓨터 수행시간	$A_2(M)$의 컴퓨터 수행시간
10	11	0.000000	0.000000
10	12	0.054945	0.054945
11	14	0.054945	0.054945
10	12	0.054945	0.000000
10	12	0.054945	0.000000
12	13	0.054945	0.000000
12	16	0.054945	0.000000
11	13	0.000000	0.054945
11	13	0.000000	0.054945
12	14	0.054945	0.000000
10	12	0.054945	0.000000
119	142	—	—

실험4. 1은 팬티엄75를 사용하였고, 가공시간을 1에서 5사이의 값을 랜덤으로 추출하여 각각의 트랜지션에 할당해서 실험하였는데,그 결과는 표4. 1에 나타나 있다.

총 100번의 실험을 시행한 결과 최소총작업수행시간이 줄어든 경우는 총 11번이었으며, 평균적으로 최소총작업수행시이 16.19% 줄어들었다. 각각의 알고리즘에 대한 컴퓨터 수행시간은 back tracking의 유무와 무관함을 알 수 있다.

4. 2 L1 알고리즘과의 비교

아직까지는 페트리네트가 FMS 스케줄링 문제에 활발하게 사용되지 않았다. 페트리네트를

이용한 스케줄링 알고리즘 중에서 가장 좋다고 생각되는 L1 알고리즘과 비교해 보고자 한다. 표4. 2에 L1 알고리즘과의 비교한 실험결과가 나타나 있으며, L1 알고리즘은 특정시점에서 하부 트랜지션이 모두 활성화된 경우에 효과적이며, 수행시간이 비교적 오래 걸린다는 단점이 있다.

〈표 4. 2〉 실험 4. 2의 결과

$A_1(M)$의 최소총작업 수행시간	L_1의 최소총작업 수행시간	$A_1(M)$의 컴퓨터 수행시간	L_1의 컴퓨터 수행시간
10	0.054945	11	0.43956
11	0.054945	9	0.43956
8	0.054945	6	0.43956
7	0.054945	5	0.43956
10	0.054945	8	0.49450
8	0.000000	6	0.43956
11	0.054945	9	0.43956
9	0.000000	10	0.43956
9	0.054945	11	0.43956
9	0.000000	13	0.49450
10	0.054945	13	0.43956
10	0.054945	11	0.43956
10	0.000000	13	0.43956
10	0.054945	13	0.43956
8	0.000000	11	0.49450
9	0.054945	8	0.43956
10	0.054945	11	0.43956
11	0.000000	13	0.43956
8	0.054945	6	0.43956
178	–	187	–

표4. 2는 가공시간을 1에서 5사이의 값으로 랜덤 추출하여 각각의 트랜지션에 할당하여 실험한 결과이다. 총 300번의 실험을 시행한 결과 $A_1(M)$ 알고리즘의 최소총작업수행시간이 L1 알고리즘의 최소총작업수행시간보다 줄어든 경우는 11번이 있었으며, L1 알고리즘보다 늘어난 경우도 8번이 있었다. 전체적으로 최소총작업수행시간이 5% 향상되었다. $A_1(M)$ 알고리즘의 평균 컴퓨터 수행시간은 0.037594 이며 L1 알고리즘의 평균 컴퓨터 수행시간은 0.4482356이다.

5. 결 론

본 연구에서 제시한 알고리즘은 모델링기법 중의 하나인 페트리네트를 사용하였다. 이러한 페트리네트는 이산사건의 성질을 정확하게 모델링하는 것이 가능하며, 시스템정지에 대한 예방을 보장하므로 분석이 용이하다는 장점이 있기 때문에, FMS 의 스케줄링 문제에 적합하다.

본 연구에서는 $A_1(M)$ 알고리즘을 제시하였는데, 페트레네트를 이용하여 시스템을 모델링하는 경우에 작성해야하는 도달가능 그래프를 부분적으로 작성할 수 있다는 특징과 이용할 수 있는 기계가 없는 경우에 시스템의 작동이 지연되는 것을 막기위해 이전의 상태로 되돌아가서 스케줄을 변경하는 back tracking을 한다는 특징이 있다. 이러한 back tracking의 유용함은 4장의 실험을 통하여 확인하였다. back tracking을 하는 경우가 back tracking을 하지 않는 경우보다 최소총작업수행시간이 16. 19% 줄어들었고, Dicesare와 Lee [3]에 의해 개발

된 L1 알고리즘과 비교한 결과 최소총작업수행 시간이 5% 줄어들었다.

앞으로는 $A_1(M)$ 알고리즘을 작업의 수와 기계의 수가 많은 경우에 대해서도 적용해볼 필요가 있다고 생각되며, back tracking 하는 시기와 back tracking 되어지는 위치에 대한 연구가 필요하다고 생각된다. 또한, FMS 스케줄링 문제도 Simulated Annealing 처럼 여러 가지 경우를 확률적으로 고려하는 방법을 적용해 볼 필요가 있다고 생각된다.

참 고 문 헌

[1] Petri, C. A., "Communication for Automation ", Doctoral Dissertion, University of Bonn, Bonn, West Germany, 1962

[2] Matsuura, H., H. Tsubone, and M. Kanezashi, "Sequencing, dispatching, and switching in a dynamic manufacturing environment," Int. J. of Prod. Res., Vol. 31, No. 7, pp. 1671-1688, 1993

[3] Dicesare, F. and D. Y. Lee, "FMS scheduling using Petri nets and heuristic search," Proceedings of the 1992 IEEE International Conference on Robotics and Automation, Nice, France, May 1992, pp. 1057-1062, 1992

[4] Chan, C. C. and H. P. Wang, "Design and development of a stochastic high-level Petri net system for FMS performance evaluation," Int. J. of Prod. Res., Vol. 31, No. 10, pp. 2415-2439, 1993

[5] Rodammer, F. and J. K. White, "A Recent Survey of Production Scheduling," IEEE Trans. Syst. Man Cybern., Vol. 18, No. 6, pp. 841-851, 1988

[6] Huang, H. P. and P. C. Chang, "Specification, modelling and control of a flexible manufacturing cell," Int. J. of Prod. Res., Vol. 30, No. 11, pp. 2515-2543, 1992

[7] Jothishankar, M. C. and H. P. Wang, "Determination of Optimal Number of Kanbans Using Stochastic Petri Nets," Journal of Manufacturing Systems, Vol. 11, No. 6, pp. 449-461, 1991

[8] Benarieh, D. and I. Miron, "Concurrent Modeling and Simulation of Reactive Manufacturing Systems Using Petri Nets," Computers ind. Engng., Vol. 20, No. 1, pp. 45-58, 1991

[9] Viswandham, N. and Y. Narahari, "Stochastic Petri net models for performance evaluation of automated manufacturing systems," Information and Decision Technologies, Vol. 14, pp 125-142, 1988

[10] Dubois, D. and Stecke, K. E., "Using Petri nets to represent production processes," Proceedings of IEEE Conference on Decision ans Control, pp. 1062-1067, 1983

[11] Jensen, K., "Colored Petri Nets and the Invariant Method," Theoretical Computer Science 14, pp. 317-336, 1981

[12] Ciardo, C. and C. Lindemann, "A Ch-

aracterization of the Stochastic Process Underlying a Stochastic Petri Net," IEEE Transactions on software engineering, Vol. 20, No. 7, pp. 506-515, 1994

[13] Boucherie, R. J., "A Characterization of Independence for Competing Markov Chains with Applications to Stochastic Petri Nets," IEEE Transactions on software engineering, Vol. 20, No. 7, pp. 536-544, 1994

[14] Teng, S. H. and J. Zhang, "A Petri net-based decomposition approach in modeling of manufacturing systems," Int. J. of Prod. Res., Vol. 31, No. 6, pp. 1423-1439, 1993

[15] Lin, C. and D. C. Marinescu, "Stochastic High-Level Petri Nets and Applications," IEEE Transactions on computers, Vol. 37, No. 7, pp. 815-825, 1988

[16] Cossins, R. and P. Ferreira, "Celeritas : a coloured Petri net approach to simulation and control of flexible manufacturing systems," Int. J. of Prod. Res., Vol. 30, No. 8, pp. 1925-1956, 1992

[17] Reddy, C. E., O. V. Krishnaiah, and D. Chaudhuri, "A Petri net based approach for analysing tool management issues in FMS," Int. J. of Prod. Res., Vol. 30, No. 6, pp. 1427-1446, 1992

[18] Commoner, F., A. W. Holf, S. Even and A. Pnueli, "Marked Directed Graphs," J. Comp. Syst. Sci, Vol. 5, 1971

[19] Baker, K. R., Introduction to Sequencing and Scheduling, John Wiley & Sons, New York, 1974

[20] Viswandham, N. and Y. Narahari, Performance Modeling of Automated Manufacturing Systems, Prentice-Hall, 1992

[21] Peterson, J. L., Petri Net Theory and The Modeling of System, Prentice-Hall, 1981