타임 패트리넷 기반의 분할 알고리즘을 이용한 스케쥴러 설계

송유진*, 이종근*

Design of the Scheduler using the Division Algorithm Based on the Time Petri net

Yu-Jin Song, Jong-Kun Lee

Abstract

In this study, we propose a scheduling analysis method of the Flexible management system using the transitive matrix.

The Scheduling problem is a combination-optimization problem basically, and a complexity is increased exponentially for a size of the problem. To reduce an increase of a complexity, we define that the basic unit of concurrency (short BUC) is a set of control flows based on behavioral properties in the net. And we propose an algorithm to divide original system into some BUC.

To sum up, we divide a petri net model of the Flexible management system into the basic unit of concurrency through the division algorithm using the transitive matrix. Then we apply it to the division-scheduling algorithm to find an efficient scheduling. Finally, we verify its efficiency with an example.

Key Words: BUC, Petri Nets, Transitive Matrix, Division algorithm, Flexible management System

* 창원대학교 컴퓨터공학과

1. 서론

유연 생산 시스템이나 다른 시스템들에서 주요한 과제 중의 하나는 경제성을 추구하는 최적화의 작업 환경을 구축할 수 있는 스케쥴링을 구축하는 것이다. 이러한 스케쥴링 문제는 주로 시뮬레이션[1]이나 선형계획법[2], 대기이론[3]을 이용하여 해결이 되어진다. 시뮬레이션은 현실적으로 정확한 해와 세세한 정보를 제공하지만 계산 시간과 컴퓨터 메모리를 많이 요구하며 적용 대상이 변하는 경우 일반성이 떨어진다. 선형계획법, 특히 정수 계획법(Integer programming)은 합리적인 시간내에 최적해에 가까운 해를 구해내지만 일단 해가 적합하지 않은 경우가 존재하여 이에 대해 고려해야 하고, 유연 생산 시스템 환경에 나타나는 유연한 라우팅이나 공유된 자원 등을 모델링 하기가 어렵다. 대기이론은 어느 정도 정확한 해를 빠른 시간 내에 구해 주지만 이론적인 가정이 실제 유연 생산 시스템과 맞지 않고 주어진 시스템의 성능을 평가하는 모델이기 때문에 주로 설계 단계에서 이루어진다.

한편 이러한 스케쥴링 문제의 연구를 위하여 많은 학자들이 패트리넷을 이용하여 순환 스케쥴링 알고리즘을 제시하였었다[4-13].

이 연구에서는 패트리넷 모델의 추이적 행렬식을 이용해 자원 공유의 기기를 중심으로 여러 개의 서브넷으로 분리하여서 각각의 서브넷을 분석하여 스케쥴링을 얻은 후 이를 종합화 하고자 하며, 이를 위해 타임 패트리넷과 추이적 행렬을 이용한 병행적 기본 단위(BUC : basic Unit of Concurrency)로 분할하여 유연 생산 시스템의 스케쥴링 알고리즘을 제안한다.

이 연구의 구성은 다음과 같다.

먼저, 2장에서는 기존의 스케쥴링 방식에 대한 관련 연구를 소개하고, 3장에서 연구의 기본적인 이론이 되는 타임 패트리넷과 추이적 행렬, 그리고 병행적 기본 단위인 BUC에 대해 설명하고, 4장에서 추이적 행렬을 이용한 BUC 단위의 분할 알고리즘을 제시하며, 5장에서는 분할 알고리즘을 이용한 스케쥴링 알고리즘을 제안한다. 한편, 6장에서 사례 연구를 통해 제안한 알고리즘을 검증하고, 7장의 결론을 맺는다.

2. 관련 연구

이번 장에서는 기존의 패트리넷을 이용한 스케쥴링 분석 방법을 소개하고자 한다.

2.1 Hillion 알고리즘[5]

Hillion 알고리즘은 각 Job의 프로세스에서 가장 최적 시간의 프로세스를 선정하여 일정을 확장케 하는 알고리즘이다.

r_j는 프로세스 j에 남아 있는 사이클 시간의 수이다. At_j는 프로세스 j의 가능한 시간이다. $At_j = (1 + r_j)^* CT$-{프로세스 j에서 최후로 스케쥴된 작업의 마지막 시간}

$Ro_j = \sum_{k \in RemainingOperation} D(t_{jk})$는 프로세스 j에서 남아 있는 작업 시간의 합이며, 이를 프로세스 j의 남아 있는 작업 부하라 한다.

비용함수의 기준 척도는 $d_j = 1 - \left(\frac{Ro_j}{At_j}\right)$이다.

2.2 Korbaa 알고리즘[10]

Korbaa 알고리즘은 각각의 작업 순서를 스케쥴링 하는 대신에 같은 형태의 치차를 사용하는 작업 공정을 재그룹하여 스케쥴링 하며, 다음과 같이 정의된다.

$$P_i = \sum_{palletstype_i} \left\lceil \frac{\sum_{O.S.tobecarriedby_i} operatingtime}{CT} \right\rceil$$

여기서,
$\lceil x \rceil$: x와 같거나 그 이상의 수 중에서 최초의 정수를 말한다.
CT : 모든 작업 공정에 대한 $C(\gamma)$값들 중 최대값
$C(\gamma) = \frac{\mu(\gamma)}{M(\gamma)}$: γ의 사이클 시간으로서 다음과 같이 구해진다.
$\mu(\gamma) = \sum_{\forall t \in \gamma} D(t)$: γ의 모든 트랜지션의 합
$M(\gamma)$: $M_0(\gamma)$와도 같이 쓸 수 있으며, γ에 있는 토큰들의 수를 말한다.

3. 타임 패트리넷과 추이적 행렬

3.1 타임 패트리넷[16]

시간 개념을 도입한 마크드 패트리넷을 타임 패트리넷이라고 한다.
[정의 3.1] 타임 패트리넷(TPN)
$TPN=(P, T, E, S, M_0, \tau)$
여기에서,
$P=\{p_1, p_2, \ldots, p_n\}$:플레이스의 유한집합 $(n \geq 0)$
$T=\{t_1, t_2, \ldots, t_n\}$:트랜지션의 유한집합 $(m \geq 0)$
$P \cap T = \emptyset$
$E: P \times T \rightarrow N$, E는 입력 함수
$S: T \times P \rightarrow N$, S는 출력 함수
$P \neq 0$, $T \neq 0$
$M_0 \in M=\{M | M: P \rightarrow N\}$, M_0는 초기 토큰 상태
τ는 시간 함수 : $\tau \rightarrow ((f), f$는 실수, $f \geq 0)$
N : 음수를 제외한 정수의 집합(양의 정수의 집합)

3.2 추이적 행렬[15,17]

추이적 행렬이란 플레이스와 트랜지션간의 관계를 행렬로 표시함으로써 초기 토큰을 가진 플레이스를 통하여 마킹의 흐름을 파악할 수 있는 행렬이다.
[정의 3.2] 표식화 플레이스 기반 추이적 행렬
L_{BP}는 표식화 플레이스 기반 추이적 행렬이라고 하며 다음과 같이 정의된다.
$L_{BP}=B^- diag(t_1, t_2, t_3, \ldots, T_n)(B^+)^T$.....(식 3.1)
여기에서, B^-와 B^+는 각각 입력과 출력함수에 대한 행렬이다.
[정의 3.3] 가중적 플레이스 기반 추이적 행렬
L_{BP}^*는 $m \times m$가중적 플레이스 기반 추이적 행렬이라고 정의한다. 만약 트랜지션 t_k가 L_{BP}의 같은 열에 s번 나타난다면 L_{BP}에 있는 t_k를 L_{BP}^*에서는 t_k/s로 표시한다.

3.3 병행적 기본 단위(BUC)[17]

병행적 기본 단위는 모델에 내재한 구조적인 병행성을 기반으로 독립적으로 수행될 수 있는 제어 흐름들을 말한다. 패트리넷 모델에서 제어 흐름은 토큰의 흐름으로 표현된다. 병행적 기본 단위에서는 하나의 토큰이 항상 존재하는 상태 불변의 특별한 성질을 가진다. 즉, 병행적 기본 단위는 토큰 하나에 대한 상태 불변이므로 제어는 하나만 존재하며 또한 제어의 흐름이 항상 상태 불변 내에 존재하게 된다.
[정의 3.4] 병행적 기본 단위로 분할
병행적인 기본단위 (BUC)로 분할하기 위해서는 해당 플레이스의 행 방향에 있는 모든 트랜지션에 대해서, 다음의 조건을 만족해야 한다.
$\sum(t_{k_i}/d_i) \geq 1$.....(식 3.2)
여기서, t_{k_i}는 행 방향에 있는 같은 이름을 가진 트랜지션을 지칭하며, d_i는 같은 트랜지션의 동일한 분모 값을 나타낸다.

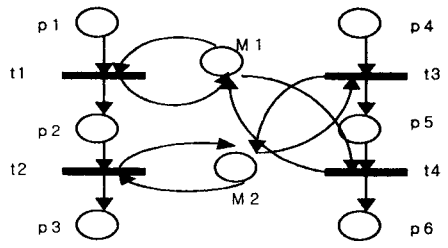
4. 추이적 행렬을 이용한 BUC 단위의 분할 알고리즘

BUC 단위의 서브넷 생성은 다음과 같은 알

고리즘[18]을 통하여 이루어진다.

```
char Psubnet[ ], Tsubnet[ ];
void search() {
 for (플레이스의 행 테이블에 트랜지션이 존재) {
  트랜지션을 읽어 온다;
  while (같은 행 테이블에 트랜지션이 존재) {
   Tsubnet[ ] 배열에 트랜지션을 넣는다;
    if (Psubnet[ ]에 그 트랜지션의 행방향의 플레이스
      가 존재하지 않는다면) {
       Psubnet[ ] 배열에 그 트랜지션의 행방향의 플레
       이스를 넣는다.;
    } /* if */
    행 테이블의 다른 트랜지션을 읽어 온다.;
  } /* while */
  다음 행 테이블로 옮긴다.;
 } /* for */
 for (Psubnet[ ] 배열에 플레이스가 존재) {
    Psubnet[ ]에 있는 플레이스의 행방향의 트랜지션을
    읽어 온다.;
    Tsubnet[ ] 배열에 트랜지션을 넣는다.;
 } /* for */
} /* search() */
main() {
 열의 플레이스와 테이블의 값을 읽어 온다.;
 for (테이블의 열에 플레이스가 존재) {
  열의 플레이스를 읽어 온다.;
  if (열의 플레이스가 초기 마킹 플레이스) {
    Psubnet[ ] 배열에 그 플레이스를 넣는다.;
    search();
  } /* if */
  if ( Σ(t_k/d≠1)) {
  /*  Tsubnet[ ] 배열에 있는 트랜지션이다. */
  트랜지션()의 행의 플레이스를 읽어 온다.;
  search();
  } /* if */
  printf (Psubnet[ ], Tsubnet[ ]);
 } /* for */
} /* main */
```

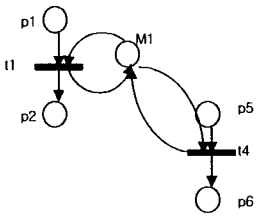
이러한 알고리즘을 간단한 예제에 적용하면 다음과 같다.



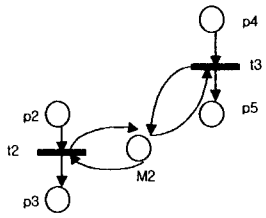
<Fig 4.1> The Petri Net Model for BUC

<Table 4.1> $L_{BP}^{*}$of the fig 4.1

	p1	p2	p3	p4	p5	p6	M1	M2
	0	t1/2	0	0	0	0	t1/2	0
	0	0	t2/2	0	0	0	0	t2/2
	0	0	0	0	0	0	0	0
	0	0	0	0	t3/2	o	o	t3/2
L^{*}_{BP} =	0	0	0	0	0	t4/2	t4/2	0
	0	0	0	0	0	0	0	0
	0	t1/2	0	0	0	t4/2	t1/2 t4/2	0
	0	0	t2/2	0	t3/2	0	0	t2/2 t3/2



<Fig 4.2> BUC of M1



<Fig 4.3> BUC of M2

4.1 분할 알고리즘의 검증

전체 패트리넷 N과 분할 알고리즘에 의하여 분할되어진 서브넷들의 집합이 서로의 부분집합이 되면 상등의 법칙에 의해 두 집합이 서로 같음을 알 수 있다. 이를 이용해 분할 알고리즘을 검증한다.

[정리 4.1] 통합

$BUC_i=\{P_i, T_i, I_i, O_i, M_i\}$ $(i=1, \ldots, n)$는 패트리넷 $N=\{P, T, I, O, M\}$을 병행적 기본 단위로 나눈 각각의 서브넷의 집합이라 하면,

i.e. $\{\forall P_i \in P, \forall T_i \in T, \forall I_i \in I, \forall O_i \in O, \forall M_i \in M\}$

BUC_i의 통합은 다음과 같다.

$BUC_1=\{P_1, T_1, I_1, O_1, M_1\}$,

$BUC_2=\{P_2, T_2, I_2, O_2, M_2\}$, ...

$BUC_n=\{P_n, T_n, I_n, O_n, M_n\}$일 경우,

$BUC_1 \cup BUC_2 \cup \ldots \cup BUC_n =$

$((P_1 \cup .. \cup P_n), (T_1 \cup .. \cup T_n), (I_1 \cup .. \cup I_n),$

$(O_1 \cup .. \cup O_n), (M_1 \cup .. \cup M_n))$

$\bigcup_{i=1}^{n} BUC_i = N(P, T, I, O, M)$이다.

[증명]

$N=\{P, T, I, O, M\}$을 패트리넷이라 하고, N을

분할 알고리즘에 의해 분할하여 얻은 서브넷 $BUC_i=\{P_i, T_i, I_i, O_i, M_i\}$ $(i=1,\ldots,n)$이라 하면, 분할된 서브넷의 $\bigcup_{i=1}^{n} BUC_i$는 분할 알고리즘의 정의에 의하여 $N\subseteq \bigcup_{i=1}^{n} BUC_i$이고,

또한 $\bigcup_{i=1}^{n} BUC_i \subseteq N$이면

상등의 법칙에 의해 $N= \bigcup_{i=1}^{n} BUC_i$이다.

(1) $N\subseteq \bigcup_{i=1}^{n} BUC_i$의 경우

분할 알고리즘에서 얻은 BUC들은 패트리넷 N의 모든 원소들의 집합에서 새로 생성되거나 제거되는 원소 없이 구성되어진 서브넷이므로, BUC의 통합인 $\bigcup_{i=1}^{n} BUC_i$은 $N\subseteq \bigcup_{i=1}^{n} BUC_i$이다.

(2) $\bigcup_{i=1}^{n} BUC_i \subseteq N$의 경우

만일 BUC_i에서,

$p_i\in P_i$, $t_i\in T_i$이 면 , $\exists p_i\in \bigcup_{i=1}^{n} P_i$, $\exists t_i\in \bigcup_{i=1}^{n} T_i$이며

$p_i\in \cdot t_i$이면, $p_i\in \bigcup_{i=1}^{n} (\cdot T_i)$이다.

분할 알고리즘에 의해 얻은 BUC_i는 모든 플레이스와 트랜지션의 삭제나 추가가 없으므로, 각 BUC_i의 플레이스의 통합과 트랜지션의 통합은 $\bigcup_{i=1}^{n} P_i\subseteq P$, $\bigcup_{i=1}^{n} T_i\subseteq T$이므로,

$p_i\in \bigcup_{i=1}^{n} (\cdot T_i) \in \cdot T$이다.

따라서, $p_i\in \cdot t_i$이면, $p_i\in \cdot t\in T$이고,

$p_i\in t_i \cdot$ 이면, $p_i\in t\cdot \in T$이다.

그러므로, $\bigcup_{i=1}^{n} BUC_i \subseteq N$이다.

이를 종합하면, $N\subseteq \bigcup_{i=1}^{n} BUC_i$, $\bigcup_{i=1}^{n} BUC_i \subseteq N$

이므로 $N= \bigcup_{i=1}^{n} BUC_i$이다.

[정리 4.2]

패트리넷 N에서 분할 알고리즘으로 얻은 $\bigcup_{i=1}^{n} BUC_i$는 N이 가지고 있는 성질(즉, 도달가능성, 생동성, 유한성 등)과 일치한다.

[증명]

정리 4.1에 의하여 $N= \bigcup_{i=1}^{n} BUC_i$이므로 기존의 성질에 변동이 없다.

5. 분할 알고리즘을 이용한 스케줄링 알고리즘

유연 생산 시스템에서의 스케줄링 분석에 대하여 먼저, 분할 알고리즘을 이용하여 모델을 분할 한 뒤, 분할된 모델을 이용하여 서브넷별로 효율적인 스케줄을 찾아내는 분할 스케줄링 알고리즘을 제안하고자 한다.

스케줄을 구하기 위해서는 먼저 비효율적인 스케줄들을 제거하는 과정이 필요하다. 따라서 스케줄링의 고려대상에서 제거해야 할 비효율적인 스케줄들, 다시 말해서 동시 수행이 가능한 모델인데 굳이 동시 수행되지 않는 스케줄들을 구하는 것은 비효율적인 스케줄이라고 할 수 있다. 이러한 비효율적인 스케줄들을 추출해내는 방법들의 논의가 먼저 필요하다.

서브넷 중에서 동시 수행이 가능한 서브넷이 있는 경우, 다음의 조건들은 비효율적이므로 고려 대상에서 제외한다. 먼저, 서브넷1과 서브넷2가 동시 수행되는 서브넷이라고 할 때, 각 서브넷들의 플레이스를 다음과 같이 정의한다.

서브넷1의 입력 플레이스 : $In_{sub1}[i]$

서브넷2의 입력 플레이스 : $In_{sub2}[i]$

서브넷1의 출력 플레이스 : $Out_{sub1}[i]$

서브넷2의 출력 플레이스 : $Out_{sub2}[i]$

[조건 5.1] 각 테이블의 입력 플레이스가 같은 위치에 있는 것은 동시 수행 될 수 없다. 이는 같은 플레이스가 동시에 다른 곳으로 입력될 수 없기 때문이다.

$In_{sub1}[i][0]=In_{sub2}[j][0]$

[조건 5.2] 한 서브넷에서 진행중인 플레이스가 출력되지 않은 상태에서 다른 서브넷의 입

력 플레이스로 사용되는 순서는 동시 수행이 불가능하다.

[조건 5.3] 한 서브넷의 제일 마지막 순서의 출력 플레이스를 다른 서브넷의 첫 입력 플레이스로 쓰이는 경우 동시 수행이라 볼 수 없다.

$Out_{sub1}[i][n-1]=In_{sub2}[j][0]$

또한, 동시 수행의 경우가 아니면, 다음조건을 만족하여야 효율적인 스케줄을 얻을 수 있다.

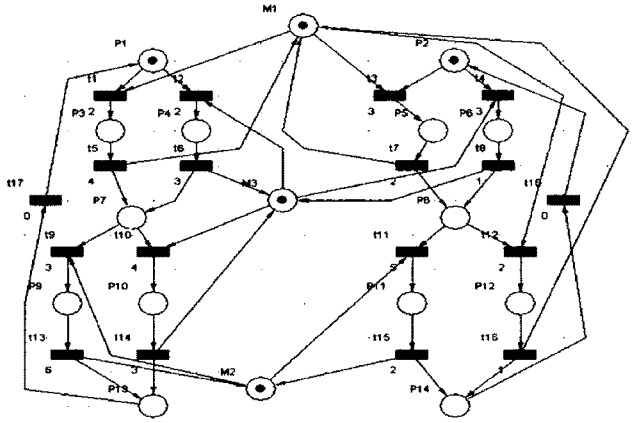
[조건 5.4] 한 테이블의 출력 플레이스와 다른 테이블의 입력 플레이스가 같은 경우, 출력 플레이스가 있는 테이블이 우선한다. 이는 출력 플레이스로 명시되어 있다는 것은 그 플레이스가 수행될 때까지 자원을 가지고 있다는 뜻이므로 그 자원이 필요한 입력 플레이스로 표시된 테이블은 그 자원을 내보낼 때까지 기다려야 한다.

위의 조건 5.1, 조건 5.2, 조건 5.3, 조건 5.4들을 적용하여 스케줄링을 찾아내는 알고리즘을 작성하면 다음과 같다.

패트리넷 모델의 추이적 행렬식 L_{BP}^*를 구한다.
L_{BP}^*에 분할 알고리즘 적용, 모델을 분할한다.
분할된 서브넷의 처리 순서를 정한다.
분할된 서브넷 각각의 sequence 테이블을 만든다.
동시 수행하는 서브넷들을 추출한다.
do {
for (i=0; i++; i<n) {
for(j=0; j++; j<m) {
for (k=0; k++; k<h) {
subnet1[i][j]와 subnet2[j][k]를 비교한다.;
If (조건[4.1][4.2][4.3]에 부합하면) {
해당 서브넷 테이블들을 제외한다.; } } }
서브넷 테이블(subnet1[i], subnet2[j])를 얻는다.
}while(동시 수행하는 서브넷 조합들이 있는 동안)
서브넷 조합들의 스케줄링 시간을 계산하여 테이블로 작성한다.
동시 수행이 아닌 서브넷들을 테이블의 해당 위치에 끼워넣는다
스케줄링을 얻는다.

6. 사례 연구

이번 장에서는 앞 장에서 제시한 스케줄링 분석 방법을 Job-Shop 사례에 적용한다.



<Fig 6.1> The Petri Net Model of the FMS

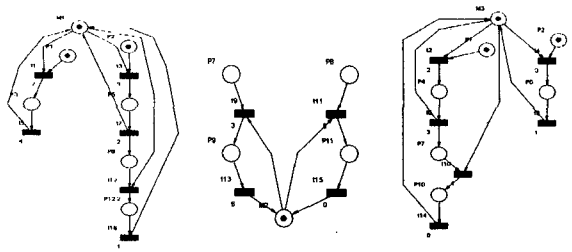
그림 6.1은 유연 생산 시스템을 패트리넷으로 모델링 한 것이다. 패트리넷 모델에서 P1과 P2는 각각 Job1과 Job2를 표현한 것이며, M1, M2, M3은 각각 Machine 1,2,3을 표현한 것이다. 그림 6.1의 패트리넷 모델을 추이적 행렬로 나타내면 표 6.1과 같다.

추이적 행렬의 플레이스 순서는 P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, M1, M2, M3이다.

<Table 6.1> $L_{BP}^*$of the fig 6.1

$$L_{BP}^* = \begin{bmatrix}
0 & 0 & t_1/2 & t_2/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & t_3/2 & t_4/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & t_5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_5 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & t_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_6 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & t_7 & 0 & 0 & 0 & 0 & 0 & 0 & t_7 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & t_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_8 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_9/2 & t_{10}/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_{11}/2 & t_{12}/2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_{13} & 0 & 0 & t_{13} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_{14} & 0 & 0 & 0 & t_{14} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_{15} & 0 & t_{15} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_{16} & t_{16} & 0 & 0 \\
t_{17} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & t_{18} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & t_1/2 & 0 & t_3/2 & 0 & 0 & 0 & 0 & 0 & 0 & t_{12}/2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_9/2 & 0 & t_{11}/2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & t_2/2 & 0 & t_4/2 & 0 & 0 & 0 & t_{10}/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

위의 추이적 행렬을 이용하여 앞 장에서 제안한 분할 알고리즘을 적용하면 다음 그림 6.2와 같이 세 개의 서브넷으로 분할되며, 이들의 구성은 표 6.2에 있다.



(a) subnet1 (b) subnet2 (c) subnet3

<Fig 6.2> Subnet of fig 6.1

<Table 6.2> Subnet Composition of fig 6.1

서브넷	플레이스와 트랜지션
서브넷1	M1-P1-P2-P3-P5-P8-P12, t1-t3-t5-t7-t12-t16
서브넷2	M2-P7-P8-P9-P11, t9-t11-t13-t15
서브넷3	M3-P1-P2-P4-P6-P7-P10, t2-t4-t6-t8-t10-t14

그림 6.1에 나타낸 패트리넷 모델의 분할을 통한 스케줄링 분석 과정은 다음과 같다.

6.1 서브넷의 스케줄링 테이블

분할된 각각의 서브넷에서 얻을 수 있는 스케줄링은 입/출력 플레이스, 점화순서와 서브넷의 입/출력 플레이스와 배열과의 관계로 표시할 수가 있다.

6.1.1 서브넷 1(sub1) 스케줄링 테이블

그림 6.2(a)의 서브넷 1은 총 3개의 스케줄링 순서가 나오게 된다.

T1→T5→T3→T7→T12→T16,
T3→T7→T12→T16→T1→T5,
T3→T7→T1→T5→T12→T16,

이러한 스케줄링 순서에 맞추어 스케줄링 테이블을 작성하면 표 6.3, 표 6.4, 표 6.5와 같다.

<Table 6.3> fig 6.2(a) Scheduling Table (1)

Sub1[0]	Sub1[0][0]	Sub1[0][1]	Sub1[0][2]	Sub1[0][3]	Sub1[0][4]	Sub1[0][5]
입력 플레이스	$In_{sub1}[0][0]$ =P1		$In_{sub1}[0][2]$ =P2		$In_{sub1}[0][4]$ =P8	
점화순서	T1(2)	T5(4)	T3(3)	T7(2)	T12(2)	T16(1)
출력 플레이스		$Out_{sub1}[0]$ [1]=P7		$Out_{sub1}[0]$ [3]=P8		$Out_{sub1}[0]$ [5]=P2

Subn[i][j]
i : Subn의 스케줄링 테이블의 경우의 수.
j : Subn의 스케줄링 테이블 내에서의 구분자.
$In_{subn}[i][j]$: Subn의 입력 플레이스 배열.
$Out_{subn}[i][j]$: Subn의 출력 플레이스 배열

<Table 6.4> fig 6.2(a) Scheduling Table (2)

Sub1[1]	Sub1[1][0]	Sub1[1][1]	Sub1[1][2]	Sub1[1][3]	Sub1[1][4]	Sub1[1][5]
입력플레이스	P2		P8		P1	
점화순서	T3(3)	T7(2)	T12(2)	T16(1)	T1(2)	T5(4)
출력플레이스		P8		P2		P7

<Table 6.5> fig 6.2(a) Scheduling Table (3)

Sub1[2]	Sub1[2][0]	Sub1[2][1]	Sub1[2][2]	Sub1[2][3]	Sub1[2][4]	Sub1[2][5]
입력플레이스	P2		P1		P8	
점화순서	T3(3)	T7(2)	T1(2)	T5(4)	T12(2)	T16(1)
출력플레이스		P8		P7		P2

6.1.2 서브넷 2(sub2) 스케줄링 테이블

서브넷 2의 스케줄링은 다음과 같다.

T9→T13→T11→T15,
T11→T15→T9→T13,

<Table 6.6> fig 6.2(b) Scheduling Table (1)

Sub2[0]	Sub2[0][0]	Sub2[0][1]	Sub2[0][2]	Sub2[0][3]
입력 플레이스	P7		P8	
점화 순서	T9(3)	T13(6)	T11(5)	T15(2)
출력 플레이스				P2

<Table 6.7> fig 6.2(b) Scheduling Table (2)

Sub2[1]	Sub2[1][0]	Sub2[1][1]	Sub2[1][2]	Sub2[1][3]
입력 플레이스	P7		P8	
점화 순서	T11(5)	T15(2)	T9(3)	T13(6)
출력 플레이스				P1

6.1.3 서브넷 3(sub3) 스케쥴링 테이블

서브넷 3의 스케쥴링은 다음과 같다.

T2→T6→T10→T14→T4→T8,

T2→T6→T4→T8→T10→T14,

T4→T8→T2→T6→T10→T14,

<Table 6.8> fig 6.2(c) Scheduling Table (1)

Sub3[0]	Sub3[0][0]	Sub3[0][1]	Sub3[0][2]	Sub3[0][3]	Sub3[0][4]	Sub3[0][5]
입력플레이스	P1		P7		P2	
점화순서	T2(2)	T6(3)	T10(4)	T14(3)	T4(3)	T8(1)
출력플레이스		P7		P1		P8

<Table 6.9> fig 6.2(c) Scheduling Table (2)

Sub3[1]	Sub3[1][0]	Sub3[1][1]	Sub3[1][2]	Sub3[1][3]	Sub3[1][4]	Sub3[1][5]
입력플레이스	P1		P2		P7	
점화순서	T2(2)	T6(3)	T4(3)	T8(1)	T10(4)	T14(3)
출력플레이스		P7		P8		P1

<Table 6.10> fig 6.2(c) Scheduling Table (3)

Sub3[2]	Sub3[2][0]	Sub3[2][1]	Sub3[2][2]	Sub3[2][3]	Sub3[2][4]	Sub3[2][5]
입력플레이스	P2		P1		P7	
점화순서	T4(3)	T8(1)	T2(2)	T6(3)	T10(4)	T14(3)
출력플레이스		P8		P7		P1

6.2 분할알고리즘을 이용한 스케쥴링의 효율성

6.1절에서 작성된 스케쥴링 테이블의 경우의 수를 고려할 때, 본 연구에서 제안한 분할 알고리즘을 이용한 스케쥴링 방식은 분할하기 전의 모델을 그대로 스케쥴링 하는 것보다 다음과 같은 측면에서 효율적임을 알 수 있다.

즉, 그림 6.1의 모델을 그대로 스케쥴링 할 경우, 선택플레이스인 P1, P2, P7, P8에서 각각 경우의 수가 2개씩 나타나게 되고, 이는 또한 Job1의 시작 플레이스인 P1과 Job2의 시작 플레이스인 P2에서 선택적으로 시작할 수 있으므로 이를 모두 조합하여 스케쥴링을 위한 경우의 수를 계산하면 2*2*2*2*2 = 32개의 스케쥴링 가능 조합이 생기게 된다. 그러나 분할을 통해 모델의 스케쥴링을 고려한다면 앞절에서 보여진 바와 같이 서브넷1에서 3가지, 서브넷2에서 2가지, 서브넷3에서 3가지의 경우의 수가 생기므로 모두 3*2*3 = 18개의 스케쥴링 가능 조합이 생기게 된다.

결국, 분할하여 스케쥴링에 접근하는 것이 그렇지 않은 경우보다 복잡도면에서 훨씬 효율적임을 알 수 있으며, 이러한 문제는 모델이 커지면 커질수록 그 차이가 현저히 벌어지게 될 것이다.

6.3 서브넷의 스케쥴링 처리 과정

분할된 각 서브넷들 중, 동시 수행하는 서브넷들의 효율적인 조합들을 찾아 내기 위해, 6.1절에서 나타낸 테이블들을 이용하여 조건 5.1, 조건 5.2, 조건 5.3, 조건 5.4를 적용시켜 비효율적인 스케쥴들을 제거하는 과정을 단계적으로 기술한다. 또한 비효율적인 조합들을 제거하고 난 뒤, 효율적인 스케쥴을 계산해 내는 과정을 역시 단계적으로 기술한다.

단계1) Sub1과 Sub3이 동시에 처음 실행된다. Sub2는 그 이후에 수행된다.

단계2) Sub1과 Sub3의 조합을 살펴본다.

단계3) 알고리즘에 의하여 동시 수행이 아닌 조합은 제거한다.

(1) Sub1[0]의 경우

-조건 5.1에 의하여 입력 플레이스가 P1이므로 Sub3의 P1으로 시작하는 것은 동시수행불가.

-Sub3[0] 와 Sub3[1]이, 이 경우에 해당된다.

-조건 5.2에 의하여 Sub3[2]는 사용중인 P1이

출력되지 않은 상태에서 P1을 사용하고자 시도하므로 동시수행불가.
-따라서 Sub1[0]와는 효율적인 조합이 나오지 않는다.
(2) Sub1[1]의 경우
-조건 5.1에 의하여 Sub3[2]와 동시수행불가
(3) Sub1[2]의 경우
-조건 5.1에 의하여 Sub3[2]와 동시수행불가
-조건 5.2에 의하여 Sub3[0], Sub3[1]동시수행불가
단계4) Sub1과 Sub3의 조합에서 동시 수행할 수 없는 조합을 제거하면 다음의 조합만을 고려한다.
-Sub1[1]과 Sub3[0]
-Sub1[1]과 Sub3[1]
단계5) 위의 조합에서 동시 수행 가능한 부분만을 추출하여 계산한다.

이를 스케쥴링 테이블로 나타내면 표 6.11, 표 6.12와 같다.

<Table 6.11> Scheduling Table of Sub1[1] and Sub3[0]

Sub1[1]	Sub1[1][0]	Sub1[1][1]	Sub1[1][2]	Sub1[1][3]	Sub1[1][4]	Sub1[1][5]
입력플레이스	P2		P8		P1	
점화순서	**T3(3)**	**T7(2)**	**T12(2)**	**T16(1)**	T1(2)	T5(4)
출력플레이스		P8		P2		P7
Sub3[0]	Sub3[0][0]	Sub3[0][1]	Sub3[0][2]	Sub3[0][3]	Sub3[0][4]	Sub3[0][5]
입력플레이스	P1		P7		P2	
점화순서	**T2(2)**	**T6(3)**	**T10(4)**	**T14(3)**	T4(3)	T8(1)
출력플레이스		P7		P1		P8

표 6.11의 진한 부분은 동시 수행이 가능하다. 왜냐하면 입력과 출력 플레이스가 중복되는 부분이 없으므로 서로 독립적으로 동작하기 때문이다. 그 이후의 부분은 서로의 출력 플레이스에 영향을 받으며, 자원공유 플레이스와도 영향을 받으므로 Sub1[1]과 Sub3[0]의 진한 부분이 모두 수행되어야 동작 가능하다.
-Sub1[1][0] + Sub1[1][1] + Sub1[1][2] + Sub1[1][3] = 8
-Sub3[0][0] + Sub3[0][1] + Sub3[0][2] + Sub3[0][3] = 12

그러므로 12의 시간이 흐른 후에 뒤의 부분이 동시에 동작 가능하다
그러나 뒤의 부분은 중복되는 입력 플레이스와 출력 플레이스가 없으므로 각각 동작한다.
즉, (Sub1[1][4]→Sub1[1][5])와 (Sub3[0][4]→Sub3[0][5])은 독립적으로 동작한다.
-Sub1[1][4] + Sub1[1][5] = 6
-Sub3[0][4] + Sub3[0][5] = 4

<Table 6.12> Scheduling Table of Sub1[1] and Sub3[1]

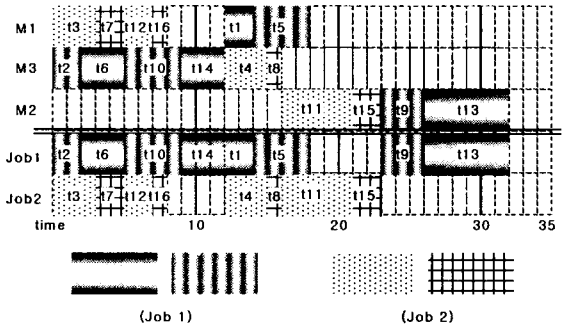
Sub1[1]	Sub1[1][0]	Sub1[1][1]	Sub1[1][2]	Sub1[1][3]	Sub1[1][4]	Sub1[1][5]
입력플레이스	P2		P8		P1	
점화순서	**T3(3)**	**T7(2)**	**T12(2)**	**T16(1)**	T1(2)	T5(4)
출력플레이스		P8		P2		P7
Sub3[1]	Sub3[1][0]	Sub3[1][1]	Sub3[1][2]	Sub3[1][3]	Sub3[1][4]	Sub3[1][5]
입력플레이스	P1		P2		P7	
점화순서	**T2(2)**	**T6(3)**	T4(3)	T8(1)	T10(4)	T14(3)
출력플레이스		P7		P8		P1

표 6.12의 진한 부분은 동시 수행이 가능하다. 왜냐하면 입력과 출력 플레이스가 중복되는 부분이 없으므로 서로 독립적으로 동작하기 때문이다.

그러나 그 이후의 부분은 입력 플레이스와 출력 플레이스가 서로 중복되어 영향을 미치므로 동시 수행이 불가능하다. 그러므로 표 6.11의 경우와 비교했을 때 비효율적이므로 채택하지 않는다.
단계6) Sub2를 고려한다.
Sub2는 입력 플레이스로 P7과 P8이 필요하므로 앞의 과정에서 먼저 그 플레이스들을 출력해 주는 것에 연결되는 것이 더 효율적이다.
따라서 Sub3[0][4] + Sub3[0][5] = 4 의 뒤에 서브넷 2가 동작하는 것이 더 효율적이다. 이 부분의 출력 플레이스는 P8이므로 서브넷 2에서 P8이 먼저 입력 플레이스로 동작하는 Sub2[1]이 선택된다.
-Sub3[0][4] + Sub3[0][5] + Sub2[1][0] + Sub2[1][1] + Sub2[1][2] + Sub2[1][3] = 20
단계7) 스케쥴링 시간 값은 12+20=32 이다.
전체의 스케쥴링 흐름을 테이블과 간트 챠트로 나타내면 표 5.13, 그림 5.3과 같다.

<Table 6.13> The Scheduling Result of the fig 6.1

Sub1 [1][0]	Sub1 [1][1]	Sub1 [1][2]	Sub1 [1][3]	Sub1 [1][4]	Sub1 [1][5]				
T3(3)	T7(2)	T12(2)	T16(1)	T1(2)	T5(4)				
Sub3 [0][0]	Sub3 [0][1]	Sub3 [0][2]	Sub3 [0][3]	Sub3 [0][4]	Sub3 [0][5]	Sub2 [1][0]	Sub2 [1][1]	Sub2 [1][2]	Sub2 [1][3]
T2(2)	T6(3)	T10(4)	T14(3)	T4(3)	T8(1)	T11(5)	T15(2)	T9(3)	T13(6)



<Fig 6.3> The Gant Chart of Scheduling

7. 결론

스케쥴링 문제는 같은 기능을 추구하지만 서로 다른 구성을 가진 유연 생산 시스템을 패트리넷으로 모델화해서 스케쥴링 분석 했을 때, 작업대의 수나 시간상 효율적인 모델을 채택하여 유연 생산 시스템을 구성할 수 있는 근간을 제공할 수 있다. 분할 스케쥴링 알고리즘은 모델을 분할하여 좀 더 작은 모델들을 가지고 분석을 시도하고, 또한 비효율적인 스케쥴들을 미리 제거함으로써, 문제의 크기에 따라 복잡도가 지수적으로 증가하게 되는 NP-hard problem을 보완하고자 시도한 알고리즘이며, 이 분석 방법은 정확한 스케쥴을 찾아냄과 동시에 많은 경우의 수를 미리 제거함으로써 복잡도를 줄일 수 있는 효과를 가져온다.

다시 말해, 추이적 행렬을 이용한 병행적 기본단위로 분할하여 스케쥴링을 분석함으로써 분할 이전의 병행적 상태를 모두 고려하는 스케쥴링의 경우의 수보다 그 경우의 수가 훨씬 줄어들게 된다. 결국 경우의 수를 줄이고 스케쥴링을 시도함으로써 스케쥴링 결과를 훨씬 빨리 얻을 수 있으며, 따라서 자동 유연 생산 시스템에 적극적으로 적용시킬 수 있게 된다.

참고문헌

[1] Y.L. Chang and R.S. Sullivan, Using SLAM to Design the Material Handling System of a Flexible Manufacturing System, International Journal of Production Research, Vol.24, pp.15-26, 1986.

[2] Y.U. Chen and R.G. Askin, A Multiobjective Evaluation of Flexible Manufacturing System Loading Heuristics, International Journal of Production Research, Vol.28, pp.895-911, 1990.

[3] Y. Cai, T. Sekiguchi, H. Tanaka, M. Hikichi and Y. Maruyama, A method for analyzing Petri net structure and adding counter-place to its incidence matrix, The Transactions of the SICE of Japan, Vol.29, No. 12, pp. 1458-1464, 1993.

[4] C. Valentin, Modeling and Analysis methods for a class of Hybrid Dynamic Systems, In: proceeding Symposium ADPM'94, pp.221-226, 1994.

[5] H.Hillion,J-M Proth, and X-L Xie, A Heuristic Algorithm for Scheduling and Sequence Job-Shop problem, In: proceeding 26th CDC,pp.612-617, 1987.

[6] H. Ohl, H. Camus, E. Castelain, and J-C. Gentina, Petri nets Modeling of Ratio-driven FMS and Implication on the WIP for Cyclic Schedules, In: Proceeding SMC'95, pp.3081-3086, 1995.

[7] H.Camus, Conduite de systmes Flexibles de Production Manufacturire par

composition de rgimes permanents cycliques : modlisation et valuation de performances l'aide des Rseaux de Petri, Thse de doctorat USTL 1,mars 1997.

[8] J.K. Lee, O. Korbaa, and J.C. Gentina, Modeling and analysis of Cycle scheduling using Petri nets unfolding, In: proceeding IEEE SMC'01, pp.2611-2616, 2001.

[9] J.K. Lee, Une mthode d'analyse d'ordinnancement des systems flxibles de production manufacturire utilisant le dpliage des rseaux de Petri, Thse de doctorat EC Paris, mars 2002.

[10] O. Korbaa, H. Camus and J-C. Gentina, FMS Cyclic Scheduling with Overlapping production cycles, In: proceeding ICATPN'97, pp.35-52, 1997.

[11] O. Korbaa, Commande cyclique des systemes flexibles de production manufacturiere a l'aide des reseaux de Petri: de la planification a l'ordonnancement des regimes transtoires, Thse de doctorat USTL 1,juillet 1998.

[12] P. Richard, Scheduling timed marked graphs with resources a serial method, In: proceeding INCOM'98, 1998.

[13] W. Zuberek and W. Kubiah, Throughput Analysis of Manufacturing Cells Using Timed Petri nets, In: Proceeding ICSYMC 1993, pp.1328-1333, 1993.

[14] W. Zuberek, Schedules of Flexible Manufacturing Cells and their Timed Colored Petri net Models, In: Proceeding ICSYMC 1995, pp.2142-2147, 1995.

[15] J. Liu, Y. Itoh, I. Miyazawa and T. Sekiguchi, A Research on Petri Net Properties using Transitive Matrix, IEEE International Conference on Systems, Man, and Cybernetics, 1999.

[16] J. Carlier, P. Chretienne and C. Girault, Modeling Scheduling Problems with Timed Petri Nets, in Advanced in Petri Nets 1984, Lecture Notes in Computer Science, G. Goes and J.Hartmanis(eds), Springer-Verlag Pub.Co., pp. 62-82, 1985.

[17] 송유진, 김종욱, 이종근, 유연생산 시스템 스케쥴링 분석을 위한 추이적 행렬을 이용한 패트리넷의 분할, 제어자동화시스템공학 논문지, 제8권, 제4호, pp. 292~298, 2002

[18] 송유진, 이종근, 추이적 행렬을 이용한 패트리넷 모델의 분석 방법에 대한 연구, 한국시뮬레이션학회 논문지, 제10권, 제1호, pp. 13~24, 2001

● 저자소개 ●

송유진(E-Mail : syj@sarim.changwon.ac.kr)
1992 창원대학교 컴퓨터공학과 졸업(학사)
1995 창원대학교 대학원 컴퓨터공학과 졸업(석사)
2003년 2월 창원대학교 컴퓨터공학과 공학박사
1995 ~ 2002 창원대학교 강사
2003 ~ 현재 창원대학교 초빙교수(IT교수요원)
관심분야 : 패트리 넷, 스케줄링분석, 성능분석, 정보보호 관련 연구

이종근(E-Mail : jklee@sarim.changwon.ac.kr)
1974 숭실대학교 전자계산학과 및 동 대학원, 공학석사
1978 고려대학교 경영대학원, 경영학 석사
1987-1990 LSI / Univ. de Montpellier II 연구원 역임
2002 LCGI/Ecole Centrale Paris 전산학, 공학박사
1983- 현재, 창원대학교 컴퓨터공학과 교수,
컴퓨터공학과장,전산소장,자연대부학장역임
관심분야 : 패트리 넷, 스케줄링분석, 성능분석, 정보보호 관련 연구