분할 스케쥴링 알고리즘에 기반한 스케쥴러의 효율성 분석

Efficiency Analysis of Scheduler based on the Division Scheduling Algorithm

송 유 진*, 이 종 근
(Yu-Jin Song and Jong-Kun Lee)

Abstract : We proposed the division algorithm that was aimed at dividing system models. It used a transitive matrix to express the relation between place and transition. And the division algorithm was applied to the scheduling problem, with the division-scheduling algorithm. The division-scheduling algorithm was able to calculate the divided subnet table. And it is able to reduce the analysis complexity. In this study, we applied the proposed division algorithm and division-scheduling algorithm to flexible manufacturing system models. We compared the efficiency and performance of the division-scheduling algorithm with the Hillion algorithm, Korbaa algorithm, and Unfolding algorithm proposed in previous researches.

Keywords : time petri net, transitive matrix, division algorithm, division-scheduling algorithm, flexible manufacturing system

I. 서론

유연 생산 시스템이나 다른 시스템들에서 주요한 과제 중의 하나는 경제성을 추구하는 최적화의 작업 환경을 구축 할 수 있는 스케쥴링을 구축하는 것이다. 이러한 연구를 위하여 많은 학자들이 패트리넷을 이용하여 순환 스케쥴링 알고리즘을 제시하였었다[1-10]. 스케쥴링의 경우 크게 세 가지의 알고리즘으로 구분이 가능하다.

Hillion[2]은 작업의 수행가능성 비율을 기초로 한 발견적 방법 모델을 제시하였고, Korbaa[7]는 기존의 작업 일정을 다시 재그룹화 하여서 작업 환경 WIP(Work In Process)를 고려한 순환 알고리즘을 제시하였다. Unfolding 알고리즘[5,6]은 다음단계의 최고대기시간까지 고려하여 분할한 뒤, unfolding이론을 적용하여 스케쥴링을 구하는 알고리즘이다.

이 연구에서는 패트리넷 모델의 추이적 행렬식을 이용해 자원 공유의 기기를 중심으로 여러 개의 서브넷으로 분리하여서 각각의 서브넷을 분석하여 스케쥴링을 얻는 분할 스케쥴링 알고리즘을 제안한다. 그리고 분할 스케쥴링 알고리즘을 유연 생산 시스템 모델에 적용시켜 스케쥴을 도출해 내고 이를 기존의 스케쥴링 분석 방법들인 Hillion 알고리즘, Korbaa알고리즘, Unfolding 알고리즘과 비교 분석하여 제안된 방법의 효율성을 검증한다.

이 연구의 구성은 다음과 같다.

먼저 2장에서는 타임 패트리넷과 추이적 행렬의 정의에 대해 설명하고, 3장에서 추이적 행렬을 이용한 BUC단위의 분할 스케쥴링 알고리즘을 제안하며, 4장에서 분할 스케쥴링 알고리즘을 사례에 적용시켜 스케쥴을 도출해 내고 5장에서 제안된 스케쥴링 알고리즘을 기존의 스케쥴링 방법들과 비교하여 그 효율성을 검증한다. 그리고 6장에서 결론을 맺는다.

II. 타임 패트리넷과 추이적 행렬

1. 타임 패트리넷

일반적인 실제 시스템에서 어떤 사건이 발생할 경우 시간 조건이 필요한 경우가 있다. 따라서 보다 정확한 모델링을 위해서는 점화시간 및 그것을 포함하는 광의의 지연시간 개념이 도입되어져야 한다. 이러한 시간 개념을 도입한 마크드 패트리넷을 타임 패트리넷[11]이라고 한다.

정의 1: 타임 패트리넷(TPN)

$TPN = (P, T, E, S, M_0, \tau)$

여기에서,

$P = \{p_1, p_2, ..., p_n\}$: 플레이스의 유한 집합 $(n \geq 0)$

$T = \{t_1, t_2, ..., t_m\}$: 트랜지션의 유한 집합 $(m \geq 0)$

$P \cap T = \phi$

$E: P \times T \rightarrow N$, E 는 입력 함수

$S: T \times P \rightarrow N$, S 는 출력 함수

$P \neq 0 \quad and \quad T \neq 0$

$M_0 \in M = \{M | M: P \rightarrow N\}$, M_0 는 초기 토큰 상태

τ 는 시간 함수 : $\tau \rightarrow ((f), f$ 는 실수, $f \geq 0)$

N : 음수를 제외한 정수의 집합(양의 정수의 집합)

2. 추이적 행렬

추이적 행렬[4]이란 플레이스와 트랜지션간의 관계를 행렬로 표시함으로써 초기 토큰을 가진 플레이스를 통하여 마킹의 흐름을 파악할 수 있는 행렬이다. 입력함수의 행렬식과 출력함수의 행렬식을 이용하여 플레이스 행렬식과 트랜지션 행렬식을 정의할 수가 있다.

정의 2: 표식화 플레이스 기반 추이적 행렬

L_{BP} 는 표식화 플레이스 기반 추이적 행렬이라고 하며 다음과 같이 정의된다.

$$L_{BP} = B^{-} diag(t_1, t_2, t_3, ..., t_n)(B^{+})^{T} \quad (1)$$

여기에서, B^{-} 와 B^{+} 는 각각 입력과 출력함수에 대한 행렬이며, $t_i (i = 1,2,3,.....n)$ 은 다음과 같다.

$|t_i| = 1$ 이면 t_i 는 점화한다.

$|t_i| = 0$ 이면 t_i 는 점화하지 않는다.

* 책임저자(Corresponding Author)
논문접수 : 2003. 4. 10., 채택확정 : 2003. 9. 5.
송유진, 이종근 : 창원대학교 컴퓨터공학과
(syj@sarim.changwon.ac.kr/jklee@ sarim.changwon.ac.kr)

L_{BP} 요소는 하나 혹은 그 이상의 트랜지션들을 통해 한 플레이스로부터 다른 플레이스로의 이동을 나타낸다.

정의 3: 가중적 플레이스 기반 추이적 행렬

L_{BP}^* 는 $m \times m$ 가중적 플레이스 기반 추이적 행렬이라고 정의한다. 만약 트랜지션 t_k 가 L_{BP} 의 같은 열에 s 번 나타난다면 L_{BP} 에 있는 t_k 를 L_{BP}^* 에서는 t_k / s 로 표시한다.

III. 추이적 행렬을 이용한 BUC단위의 분할 스케쥴링 알고리즘

1. 병행적 기본 단위

병행적 기본 단위(BUC : Basic Unit of Concurrent)는 모델에 내재한 구조적인 병행성을 기반으로 독립적으로 수행될 수 있는 제어 흐름들을 말한다. 패트리넷 모델에서 제어 흐름은 토큰의 흐름으로 표현된다. 하나의 제어 흐름은 하나의 토큰이 머무는 영역을 나타내며 트랜지션에 의해 여러 개의 토큰으로 분할되는 경우는 제어 흐름 또한 여러 개로 나뉘어진다. 그리고 하나의 제어 흐름을 이루기 위해서는 제어 토큰이 항상 기본 단위를 이루는 부분 내에 머무를 수 있어야 한다. 상태 불변의 직관적인 의미는 트랜지션의 수행과 상관없이 항상 토큰의 합이 일정하게 유지되는 플레이스들의 집합을 나타낸다. 즉, 병행적 기본 단위는 토큰 하나에 대한 상태 불변이므로 제어는 하나만 존재하며 또한 제어의 흐름이 항상 상태 불변 내에 존재하게 된다.

정의 4: 병행적 기본 단위로 분할

병행적 기본단위 (BUC)로 분할하기 위해서는 해당 플레이스의 행 방향에 있는 모든 트랜지션에 대해서, 다음의 조건을 만족하여야 한다.

$$\sum (t_{k_i} / d_i) \geq 1 \qquad (2)$$

여기서, t_{k_i} 는 행 방향에 있는 같은 이름을 가진 트랜지션을 지칭하며, d_i 는 같은 트랜지션의 동일한 분모 값을 나타낸다. 따라서, 같은 트랜지션은 모두 (2)의 조건을 만족하여야 한다. 이 조건에 의해 추출되어진 패트리넷은 병행적 기본단위로 분할되어진 것이다.

2. BUC단위의 분할 알고리즘

BUC 단위의 서브넷 생성은 분할 알고리즘을 통하여 이루어지며, 분할 알고리즘은 다음과 같다.

```
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 플레
    이스가 존재하지 않는다면) {
            Psubnet[ ] 배열에 그 트랜지션의 행방향
        의 플레이스를 넣는다.;
} /* if */
행 테이블의 다른 트랜지션을 읽어 온다.,
} /* while */
다음 행 테이블로 옮긴다.;
} /* for */
for (Psubnet[ ] 배열에 플레이스가 존재) {
        Psubnet[ ]에 있는 플레이스의 행방향의 트랜지션
    을 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 ( ∑ t_k / d ≠ 1 ) {
/* t_k Tsubnet[ ] 배열에 있는 트랜지션이다. */
트랜지션(t_k )의 행의 플레이스를 읽어 온다.;
search();
} /* if */
printf (Psubnet[ ], Tsubnet[ ]);
} /* for */
} /* main */
```

3. 분할 알고리즘을 이용한 분할 스케쥴링 알고리즘

이 절에서는, 유연 생산 시스템에서의 스케쥴링 분석에 대하여 먼저, 분할 알고리즘을 적용하여 생성된, 분할 모델들을 이용하여 서브넷별(자원공유, BUC)로 효율적인 스케쥴을 찾아내는 분할 스케쥴링 알고리즘을 제안하고자 한다.

스케쥴을 구하기 위해서는 먼저 비효율적인 스케쥴들을 제거하는 과정이 필요하다. 즉, 동시 수행이 가능한 모델인데 굳이 동시 수행되지 않는 스케쥴들을 구하는 것은 비효율적인 스케쥴이라고 할 수 있으며, 따라서 이러한 비효율적인 스케쥴들을 추출해내는 방법들의 논의가 필요하다.

이의 설명을 위해 먼저, 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 : $Out_{sub2}[i]$

동시 수행되지 않는 경우, 즉 다음 조건들과 같은 경우는 고려 대상에서 제외시킨다.

조건 1. $In_{sub1}[i][0] = In_{sub2}[j][0]$

조건 2.

```
For (i=0; i++; i<n)  {
For (j=0; j++; j<m)  {
If( In_sub1[i]!= Out_sub1[j] )
{
for(k=0; k++; k<l) {
```

```
if( In_sub1[i] == In_sub2[k] )
ErrorMsg("동시 수행이 불가능.");
}}}}
```

조건 3. $Out_{sub1}[i][n-1] = In_{sub2}[j][0]$

동시 수행되지 않는다면 다음 조건을 만족하는 스케쥴링을 설정한다.

조건 4. 한 서브넷의 출력 플레이스와 다른 서브넷의 입력 플레이스가 같은 경우, 출력 플레이스가 있는 테이블이 우선한다.

위의 조건 1, 조건 2, 조건 3, 조건 4들을 적용하여 스케쥴링을 찾아내는 알고리즘을 작성하면 다음과 같다.

패트리넷 모델의 추이적 행렬식 L_{BP}^{*} 를 구한다.

L_{BP}^{*} 를 이용한 분할 알고리즘을 적용하여 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)
{
배열subnet1[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 스케쥴링 시간을 계산하여 테이블로 작성한다.

동시 수행이 아닌 서브넷들을 테이블의 해당 위치에 끼워넣는다

스케쥴링을 얻는다.

IV. 사례연구

이 장에서는 비교 분석을 위한 예제모델[12]을 그림 1과 같이 설정한다. 이 모델의 구성은 두 가지의 작업 형태인 P1과 P2가 있고 각각 세 개의 기계 U1, M1, M2에 의해 생산된다. P1은 세 개의 작업, U1(2 t.u), M1(3 t.u), M2(2 t.u)를 포함하고 있고, P2는 두 개의 작업, M1(1 t.u), U1(2 t.u)을 포함하고 있으며, 생산범위는 고정되어 있다. 또한 각 사이클에서 3/5와 2/5의 비율 생산성을 가지게 가정하여 이를 1/5 사이클 3개와 1/5 사이클 2개의 사이클로 구분하여 모델링 할 수가 있다. 운반대는 P1과 P2의 두 종류로 가정한다. 각 부분의 작업 순서는 각각 OS1과 OS2라고 한다.

먼저 분할 알고리즘을 이용해 모델을 분할하기 위해 그림 1의 각각의 플레이스와 트랜지션에 이름을 부여하여 그림 2

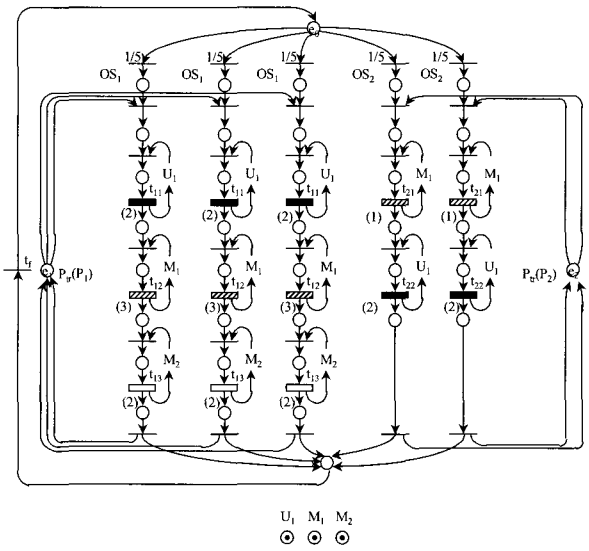


그림 1. 스케쥴링 분석을 위한 예제 모델.

Fig. 1. Example Model for scheduling analysis.

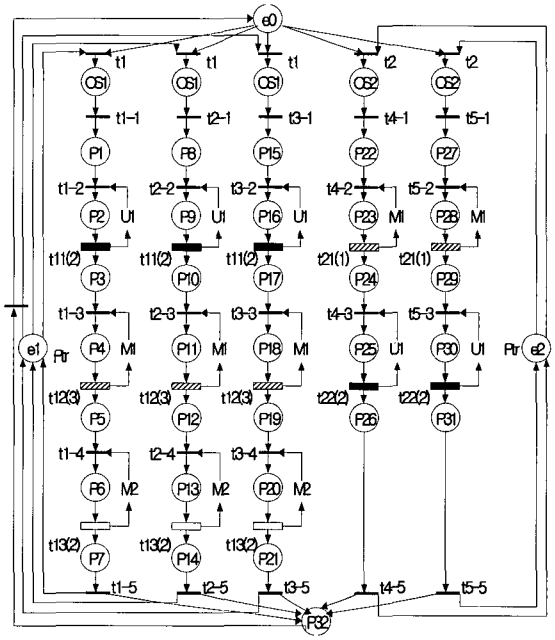


그림 2 분할 스케쥴링을 위한 예제 모델.

Fig. 2. The Example Model for the division scheduling.

에 나타내었다. 그리고 그림 2 모델의 L_{BP}^{*} 를 구했다. L_{BP}^{*} 는 지면 사정상 표1에 나누어 나타내었다.

표 1의 L_{BP}^{*} 들을 이용하여 3 개의 자원 공유 기기를 중심으로 분할 알고리즘을 이용하여 3 개의 분할 모델을 구할 수가 있다.

1. 기계 U1 서브넷의 모델과 테이블

기계 U1을 중심으로 모델을 분할하여 얻은 서브넷은 그림 3과 같다.

그림 3에서, t11의 점화에 의해 수행되는 job1을 $SubU1_{job1}$ 으로, t22의 점화에 의해 수행되는 job2를 $SubU1_{job2}$ 로 나타

표 1. 그림 4.2의 L_{BP}^{*}.

Table 1. L_{BP}^{*} of the fig 4.2.

$L_{BP}^{*}=$

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15	p16	p17	p18	p19	p20
p1	0	$\frac{t1-2}{2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p2	0	0	$\frac{t11}{3}$	0	0	0	0	0	0	$\frac{t11}{3}$	0	0	0	0	0	0	$\frac{t11}{3}$	0	0	0
p3	0	0	0	$\frac{t1-3}{2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p4	0	0	0	0	$\frac{t12}{3}$	0	0	0	0	0	0	$\frac{t12}{3}$	0	0	0	0	0	0	$\frac{t12}{3}$	0
p5	0	0	0	0	0	$\frac{t1-4}{2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p6	0	0	0	0	0	0	$\frac{t13}{3}$	0	0	0	0	0	0	$\frac{t13}{3}$	0	0	0	0	0	0
p7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p8	0	0	0	0	0	0	0	0	$\frac{t2-2}{2}$	0	0	0	0	0	0	0	0	0	0	0
p9	0	0	$\frac{t11}{3}$	0	0	0	0	0	0	$\frac{t11}{3}$	0	0	0	0	0	0	$\frac{t11}{3}$	0	0	0
p10	0	0	0	0	0	0	0	0	0	0	$\frac{t2-3}{2}$	0	0	0	0	0	0	0	0	0
p11	0	0	0	0	$\frac{t12}{3}$	0	0	0	0	0	0	$\frac{t12}{3}$	0	0	0	0	0	0	$\frac{t12}{3}$	0
p12	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{t2-4}{2}$	0	0	0	0	0	0	0
p13	0	0	0	0	0	0	$\frac{t13}{3}$	0	0	0	0	0	0	$\frac{t13}{3}$	0	0	0	0	0	0
p14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{t3-2}{2}$	0	0	0	0	0
p16	0	0	$\frac{t11}{3}$	0	0	0	0	0	0	$\frac{t11}{3}$	0	0	0	0	0	$\frac{t11}{3}$	0	0	0	0
p17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{t3-3}{2}$	0	0	0
p18	0	0	0	0	$\frac{t12}{3}$	0	0	0	0	0	0	$\frac{t12}{3}$	0	0	0	0	0	$\frac{t12}{3}$	0	0
p19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{t3-4}{2}$	0
p20	0	0	0	0	0	0	$\frac{t13}{3}$	0	0	0	0	0	0	$\frac{t13}{3}$	0	0	0	0	0	0

$L_{BP}^{*}=$

	p21	p22	p23	p24	p25	p26	p27	p28	p29	p30	p31	p32	e0	e1	e2	os1	os2	u1	m1	m2
p1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{t11}{3}$	0	0
p3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{t12}{3}$	0
p5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p6	$\frac{t13}{3}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{t13}{3}$
p7	0	0	0	0	0	0	0	0	0	0	0	0	t1-5	0	t1-5	0	0	0	0	0
p8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{t11}{3}$	0	0
p10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{t12}{3}$	0
p12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p13	$\frac{t13}{3}$	0	0	0	0	0	0	0	0	0	0	0	t2-5	0	t2-5	0	0	0	0	$\frac{t13}{3}$
p14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{t11}{3}$	0	0
p17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{t12}{3}$	0
p19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p20	$\frac{t13}{3}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{t13}{3}$

$L_{BP}^{*}=$

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15	p16	p17	p18	p19	p20
p21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e1	0	$\frac{t1-1}{2}$	0	0	0	0	0	0	$\frac{t2-1}{2}$	0	0	0	0	0	0	0	$\frac{t3-1}{2}$	0	0	0	0
e2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
os1	0	$\frac{t1-1}{2}$	0	0	0	0	0	0	$\frac{t2-1}{2}$	0	0	0	0	0	0	0	$\frac{t3-1}{2}$	0	0	0	0
os2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u1	0	$\frac{t1-2}{2}$	0	0	0	0	0	0	$\frac{t2-2}{2}$	0	0	0	0	0	0	0	$\frac{t3-2}{2}$	0	0	0	0
m1	0	0	0	$\frac{t1-3}{2}$	0	0	0	0	0	0	$\frac{t2-3}{2}$	0	0	0	0	0	0	$\frac{t3-3}{2}$	0	0
m2	0	0	0	0	0	$\frac{t1-4}{2}$	0	0	0	0	0	$\frac{t2-4}{2}$	0	0	0	0	0	0	0	$\frac{t3-4}{2}$

$L_{BP}^{*}=$

	p21	p22	p23	p24	p25	p26	p27	p28	p29	p30	p31	p32	e0	e1	e2	os1	os2	u1	m1	m2
p21	0	0	0	0	0	0	0	0	0	0	0	t3-5	0	t3-5	0	0	0	0	0	0
p22	0	0	$\frac{t4-2}{2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p23	0	0	0	$\frac{t21}{2}$	0	0	0	0	$\frac{t21}{2}$	0	0	0	0	0	0	0	0	0	$\frac{t21}{2}$	0
p24	0	0	0	0	$\frac{t4-3}{2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p25	0	0	0	0	0	$\frac{t22}{2}$	0	0	0	0	$\frac{t22}{2}$	0	0	0	0	0	0	0	$\frac{t22}{2}$	0
p26	0	0	0	0	0	0	0	0	0	0	0	t4-4	0	0	t4-4	0	0	0	0	0
p27	0	0	0	0	0	0	0	$\frac{t5-2}{2}$	0	0	0	0	0	0	0	0	0	0	0	0
p28	0	0	0	$\frac{t21}{2}$	0	0	0	0	$\frac{t21}{2}$	0	0	0	0	0	0	0	0	0	$\frac{t21}{2}$	0
p29	0	0	0	0	0	0	0	0	0	$\frac{t5-3}{2}$	0	0	0	0	0	0	0	0	0	0
p30	0	0	0	0	0	$\frac{t22}{2}$	0	0	0	0	$\frac{t22}{2}$	0	0	0	0	0	0	0	$\frac{t22}{2}$	0
p31	0	0	0	0	0	0	0	0	0	0	0	t5-4	0	0	t5-4	0	0	0	0	0
p32	0	0	0	0	0	0	0	0	0	0	0	tf	0	0	0	0	0	0	0	0
e0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e1	0	0	$\frac{t4-1}{2}$	0	0	0	0	$\frac{t5-1}{2}$	0	0	0	0	0	0	0	0	0	0	0	0
e2	0	$\frac{t4-1}{2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
os1	0	0	0	0	0	0	0	$\frac{t5-1}{2}$	0	0	0	0	0	0	0	0	0	0	0	0
os2	0	$\frac{t4-1}{2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u1	0	0	0	0	$\frac{t4-3}{2}$	0	0	0	0	$\frac{t5-3}{2}$	0	0	0	0	0	0	0	0	0	0
m1	0	0	$\frac{t4-2}{2}$	0	0	0	0	$\frac{t5-2}{2}$	0	0	0	0	0	0	0	0	0	0	0	0
m2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

표 2. 분할된 서브넷들의 플레이스와 트랜지션.

Table 2. The Place and Transition of divided Subnets.

서브넷1 (U1)	플레이스	U_1-P_1-P_2-P_3-P_8-P_9-P_{10}-P_{15}-P_{16}- P_{17}-P_{24}-P_{25}-P_{26}-P_{29}-P_{30}-P_{31}
	트랜지션	t_{1-2}-t_{2-2}-t_{3-2}-t_{4-3}-t_{5-3}-t_{11}-t_{22}
서브넷2 (M1)	플레이스	M_1-P_3-P_4-P_5-P_{10}-P_{11}-P_{12}-P_{17}- P_{18}-P_{19}-P_{22}-P_{23}-P_{24}-P_{27}-P_{29}
	트랜지션	t_{1-3}-t_{2-3}-t_{3-3}-t_{4-2}-t_{5-2}-t_{12}-t_{21}
서브넷3 (M2)	플레이스	M_2-P_5-P_6-P_7-P_{12}-P_{13}-P_{14}-P_{19}- P_{20}-P_{21}
	트랜지션	t_{1-4}-t_{2-4}-t_{3-4}-t_{13}

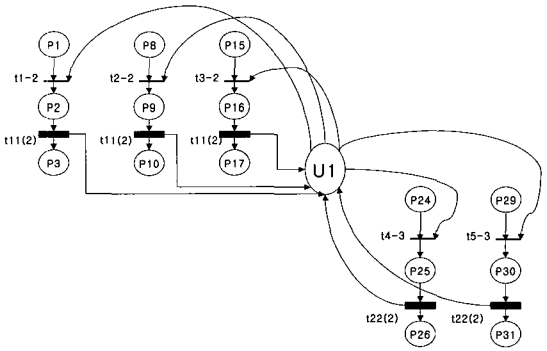


그림 3. 그림2의 서브넷 1(U1).

Fig. 3. Subnet 1(U1) of the fig2.

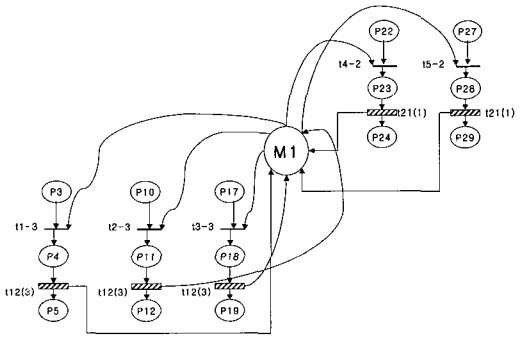


그림 4. 그림2의 서브넷 2(M1).

Fig. 4. Subnet 2(M1) of the fig2.

표 3. 그림3의 서브넷 테이블.
Table 3. Subnet Table of the fig3.

$SubU1_{job1}[0][0]$	입력 플레이스	$In_{Job1}^{SubU1}[0][0]$ =P1,P2,U1
	점화 순서	T11(2)
	출력 플레이스	$Out_{Job1}^{SubU1}[0][0]$ =P3,U1
$SubU1_{job1}[1][0]$	입력 플레이스	$In_{Job1}^{SubU1}[1][0]$ =P8,P9,U1
	점화 순서	T11(2)
	출력 플레이스	$Out_{Job1}^{SubU1}[1][0]$ =P10,U1
$SubU1_{job1}[2][0]$	입력 플레이스	$In_{Job1}^{SubU1}[2][0]$ =P15,P16,U1
	점화 순서	T11(2)
	출력 플레이스	$Out_{Job1}^{SubU1}[2][0]$ =P17,U1
$SubU1_{job2}[0][0]$	입력 플레이스	$In_{Job2}^{SubU1}[0][0]$ =P24,P25,U1
	점화 순서	T22(2)
	출력 플레이스	$Out_{Job2}^{SubU1}[0][0]$ =P26,U1
$SubU1_{job2}[1][0]$	입력 플레이스	$In_{Job2}^{SubU1}[1][0]$ =P29,P30,U1
	점화 순서	T22(2)
	출력 플레이스	$Out_{Job2}^{SubU1}[1][0]$ =P31,U1

표 4. 그림4의 서브넷 테이블.
Table 4. Subnet Table of the fig4.

$SubM1_{job1}[0][0]$	입력 플레이스	$In_{Job1}^{SubM1}[0][0]$ =P3,P4,M1
	점화 순서	T12(3)
	출력 플레이스	$Out_{Job1}^{SubM1}[0][0]$ =P5,M1
$SubM1_{job1}[1][0]$	입력 플레이스	$In_{Job1}^{SubM1}[1][0]$ =P10,P11,M1
	점화 순서	T12(3)
	출력 플레이스	$Out_{Job1}^{SubM1}[1][0]$ =P12,M1
$SubM1_{job1}[2][0]$	입력 플레이스	$In_{Job1}^{SubM1}[2][0]$ =P17,P18,M1
	점화 순서	T12(3)
	출력 플레이스	$Out_{Job1}^{SubM1}[2][0]$ =P19,M1
$SubM1_{job2}[0][0]$	입력 플레이스	$In_{Job2}^{SubM1}[0][0]$ =P22,P23,M1
	점화 순서	T21(1)
	출력 플레이스	$Out_{Job2}^{SubM1}[0][0]$ =P24,M1
$SubM1_{job2}[1][0]$	입력 플레이스	$In_{Job2}^{SubM1}[1][0]$ =P27,P28,M1
	점화 순서	T21(1)
	출력 플레이스	$Out_{Job2}^{SubM1}[1][0]$ =P29,M1

내고, job1에서 t11의 세 가지 점화 경우의 각각의 입력 플레이스, 점화 순서, 출력 플레이스를 첨자를 붙여 $SubU1_{job1}[i][j]$ 의 형태로 정의하고, job2 역시 t22의 두 가지 점화 경우의 각각의 입력 플레이스, 점화 순서, 출력 플레이스를 $SubU1_{job2}[i][j]$ 의 형태로 정의하여 나타낸다. 따라서, 기계 U1 서브넷의 각 점화 순서에 해당하는 입력 플레이스와 출력 플레이스를 테이블로 나타내면 표 3과 같다.

2. 기계 M1 서브넷의 모델과 테이블

기계 M1을 중심으로 모델을 분할하여 얻은 서브넷은 그림 4와 같고, 이를 테이블로 나타내면 표 4와 같다.

3. 기계 M2 서브넷의 모델과 테이블

기계 M2을 중심으로 모델을 분할하여 얻은 서브넷은 그림 5와 같고, 이를 테이블로 나타내면 표 5와 같다.

4. 분할 스케쥴링

분할 스케쥴링 알고리즘으로 스케쥴을 구하는 과정을 기술하면 다음과 같다.

단계1) 분할된 서브넷의 각각의 테이블들의 입력 플레이스와 출력 플레이스를 보고 조건1, 조건2, 조건3, 조건4에 의해 동시 수행 여부와 각 작업의 순서를 정한다.

먼저 각 작업대의 서로 다른 치차는 서로 겹치는 입력 플레이스와 출력 플레이스가 없으므로 동시 수행 될 수 있으나, 같은 치차 내에서는 조건 4에 의해 다음과 같은 순서를 가지게 된다.

(1) Job1의 경우 :

분할된 서브넷들을 통해 작성된 서브넷 테이블에서 보면, Job1에서의 첫번째 치차에서 기계 U1의 입력 플레이스 $In_{Job1}^{SubU1}[0][0]$ 는 P1, P2, U1 이며, 출력 플레이스 $Out_{Job1}^{SubU1}[0][0]$ 는 P3, U1이다. 그리고 Job1의 기계 M1의 입력 플레이스 $In_{Job1}^{SubM1}[0][0]$ 는 P3, P4, M1 이고, 출력 플레이스 $Out_{Job1}^{SubM1}[0][0]$ 는 P5, M1이다.

표 5. 그림5의 서브넷 테이블.
Table 5. Subnet Table of the fig5.

$SubM2_{job1}[0][0]$	입력 플레이스	$In_{Job1}^{SubM2}[0][0]$ = P5, P6, M2
	점화 순서	T13(2)
	출력 플레이스	$Out_{Job1}^{SubM2}[0][0]$ =P7, M2
$SubM2_{job1}[1][0]$	입력 플레이스	$In_{Job1}^{SubM2}[1][0]$ = P12, P13, M2
	점화 순서	T13(2)
	출력 플레이스	$Out_{Job1}^{SubM2}[1][0]$ =P14, M2
$SubM2_{job1}[2][0]$	입력 플레이스	$In_{Job1}^{SubM2}[2][0]$ = P19, P20, M2
	점화 순서	T13(2)
	출력 플레이스	$Out_{Job1}^{SubM2}[2][0]$ =P21, M2

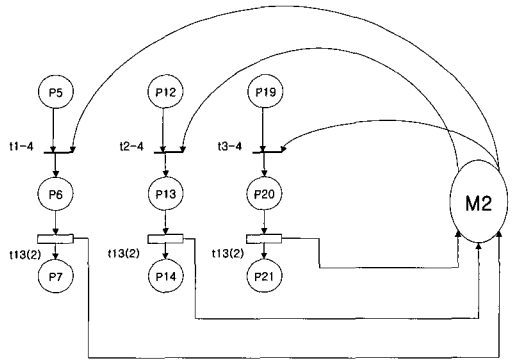


그림 5. 그림2의 서브넷 3(M2).
Fig. 5. Subnet 3(M2) of the fig2.

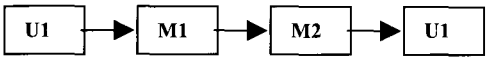


그림 6. Job1의 기계별 스케쥴 순서.
Fig. 6. Machine schedule order of the Job1.

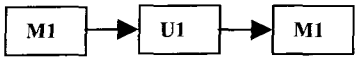


그림 7. Job2의 기계별 스케쥴 순서.
Fig. 7. Machine schedule order of the Job2.

그러므로 기계 U1에서 P3으로의 토큰의 출력이 있어야 기계 M1의 수행에 필요한 입력 플레이스 P3이 만족 될 수 있으므로, 이 작업에서는 반드시 U1의 수행 후 M1이 수행된다.

마찬가지로 기계 M2를 살펴보면 M2의 입력 플레이스 $In_{Job1}^{SubM2}[0][0]$ 는 P5, P6, M2이고, 출력 플레이스 $Out_{Job1}^{SubM2}[0][0]$ 는 P7, M2이다. 따라서 M1에서 P5로의 출력이 있어야 M2의 입력 플레이스 P5가 만족되므로 M1의 수행 후 M2가 수행되어야 한다.

그리고 이 작업은 계속 돌아가므로 M2 기계의 수행 후 다시 U1 기계의 수행으로 돌아간다. Job1에서의 두번째와 세번째 치차에서도 같은 형태로 돌아간다. 이 반복되는 작업을 그림으로 표현하면 그림 6과 같다.

(2) Job2의 경우 :

Job2에서의 경우도 Job1에서의 경우와 같이 살펴보면, 첫번째 치차에서 기계 U1의 입력 플레이스 $In_{Job2}^{SubU1}[0][0]$ 는 P24, P25, U1, 출력 플레이스 $Out_{Job2}^{SubU1}[0][0]$ 는 P26, U1이다. 또한 기계 M1의 입력 플레이스 $In_{Job2}^{SubM1}[0][0]$ 는 P22, P23, M1, 출력 플레이스 $Out_{Job2}^{SubM1}[0][0]$ 는 P24, M1이다. 기계 M2는 Job2에서의 작업이 없다.

따라서 각 기계의 입력 플레이스와 출력 플레이스에서 볼 때 기계 M1의 P24가 출력이 되어야 기계 U1의 입력 플레이스 P24의 조건이 만족 되므로 M1의 수행 후 U1이 수행된다. 그러나 이 작업도 U1의 수행 후 다시 M1으로 반복되어 돌아가므로 그림 7과 같은 그림으로 표시할 수 있다.

단계2) 각 기계 별로 분할 알고리즘을 적용하였으므로 스케쥴도 각 기계별로 작성한다. 먼저 각 기계 별로의 시간 값을 계산하여 총 시간이 가장 긴 것부터 고려한다. 이는 가장 시간 값이 긴 것이 그 스케쥴의 최적 시간 값으로 결정되기 때문이다.

$$U1 = t11(2\ t.u) + t11(2\ t.u) + t11(2\ t.u) + t22(2\ t.u) + t22(2\ t.u) = 10$$
$$M1 = t12(3\ t.u) + t12(3\ t.u) + t12(3\ t.u) + t21(1\ t.u) + t21(1\ t.u) = 11$$
$$M2 = t12(2\ t.u) + t12(2\ t.u) + t12(2\ t.u) = 6$$

위와 같은 단계들을 거쳐 이 연구에서 제안한 분할 스케쥴링 알고리즘으로 스케쥴링 분석을 한 결과 그림 8과 같은 스케쥴링이 도출된다.

결국, 최종 스케쥴링 결과는 WIP가 4개 필요하고, 대기손실시간은 17이 된다.

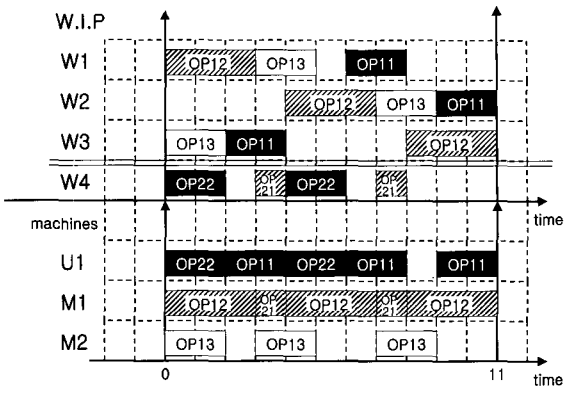


그림 8. 분할 스케쥴링 알고리즘의 스케쥴 결과.
Fig. 8. Schedule Result of the Division Scheduling Algorithm.

V. 분할 스케쥴링 알고리즘의 성능평가

이번 장에서는 기존의 스케쥴링 분석 방법(Hillion, Korbaa, Unfolding 기법)과 IV장에서 제안한 기법을 예제 모델을 통하여 비교함으로써 분할 스케쥴링 알고리즘의 장단점과 효율성을 분석하고자 한다.

1. Hillion 알고리즘

Hillion 알고리즘[2]은 각 Job의 프로세스에서 가장 최적 시간의 프로세스를 선정하여 일정을 확장케 하는 알고리즘으로 다음과 같이 정리가 가능하다.

r_j 는 프로세스 j에 대한 남아 있는 사이클 시간의 수이다.

At_j 는 프로세스 j의 가능한 시간이다.

$At_j = (1 + r_j) * CT$ -{프로세스 j에서 최후로 스케쥴된 작업의 마지막 시간}

$Ro_j = \sum_{k \in \mathrm{Re}\ mainingOperations} D(t_{jk})$ 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.

이와 같은 Hillion 알고리즘을 적용하여 그림 1에 대한 스케쥴을 계산하면 그림 9와 같다.

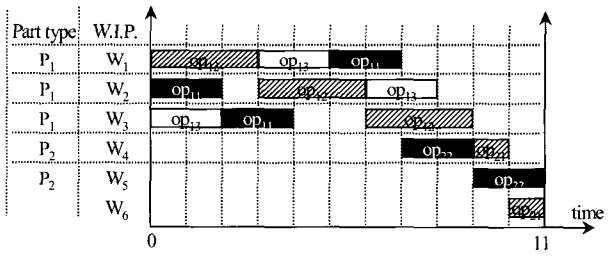


그림 9. Hillion 알고리즘의 결과.
Fig. 9. The Result of the Hillion Algorithm.

2. Korbaa 알고리즘

Korbaa 알고리즘[7]은 각각의 작업 순서를 스케쥴링 하는 대신에 같은 형태의 치차를 사용하는 작업 공정을 재그룹 하

여 스케쥴링 하며, 사이클 시간과 WIP(Work In Process)와 같은 비용함수로 이루어진다. WIP는 스케쥴을 구할 때 필요한 치차의 수를 나타내는 것으로써 다음과 같이 정의된다.

$$P_i = \sum_{pallets\ type\ i} \left\lceil \frac{\sum operating\ time_{O.S.to\ be\ carried\ by\ i}}{CT} \right\rceil$$

여기서,

$\lceil x \rceil$: x와 같거나 그 이상의 수 중에서 최초의 정수를 말한다.

CT : 모든 작업 공정에 대한 $C(\gamma)$ 값들 중 최대값.

$C(\gamma) = \frac{\mu(\gamma)}{M(\gamma)}$: γ 의 사이클 시간으로서 다음과 같이 구해진다.

$\mu(\gamma) = \sum_{\forall t \in \gamma} D(t)$: γ 의 모든 트랜지션의 합으로서 다음과 같이 구해진다.

$M(\gamma)$: $M_0(\gamma)$ 와도 같이 쓸 수 있으며, γ 에 있는 토큰들의 수를 말한다.

위의 식을 바탕으로 Korbaa 알고리즘에서 말하는 WIP는 다음과 같이 나타낸다.

$$W.I.P._{expected} = \sum_{all\ regroupings\ R_i} \left\lceil \frac{Operating\ times\ of\ R_i}{CT} \right\rceil$$

사이클의 재그룹화를 통해 Korbaa알고리즘을 적용한 그림 1의 스케쥴은 다음 그림 10과 같다.

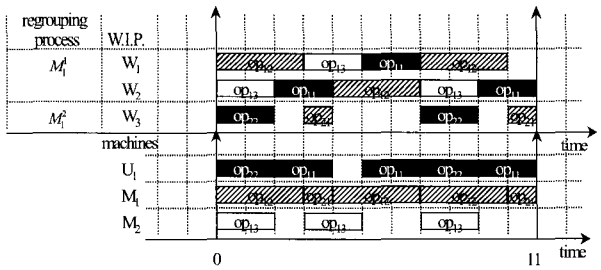


그림 10. Korbaa 알고리즘의 결과.

Fig. 10. The Result of the Korbaa Algorithm.

3. Unfolding 알고리즘

Unfolding 알고리즘[5,6]은 다음 단계의 최고 대기 시간까지 고려한 BUC 단위로 분할을 하고, Unfolding 이론을 사용하여 각각의 BUC를 분석하는 것과 같은 두 가지 단계로 스케쥴링을 구하고자 하는 알고리즘이다.

$$f(UTPN) = \underset{1 \le j \le n}{MAX} \left\{ \left(\sum_{i=1}^{j} h(t_i) \right) + g(t_j) \right\}$$

여기에서,

$h(t_i)$ 는 트랜지션 t_i 의 작업 시간

$g(t_i)$ 는 트랜지션 t_i 의 최소 대기 시간

Unfolding UTPN의 스케쥴은 함수 $f(UTPN)$ 의 최소화에 의해 얻어진다.

$$OptimizeUTPN = \min(f(UTPN_i))$$

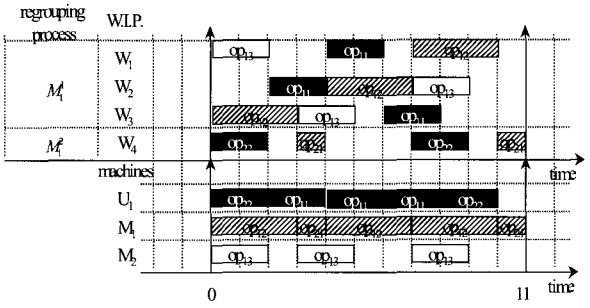
Unfolding 방법은 기계의 작업 시간의 산출에 자원 배분의 수를 말하는 토큰의 수를 고려한다. 이러한 기계의 작업 시간 차수를 $d(m_i)$ 라고 하며 이를 식으로 나타내면 다음과 같다.

$d(m_i) = \frac{\varphi(m_i)}{\gamma(m_i)}$ 여기에서,

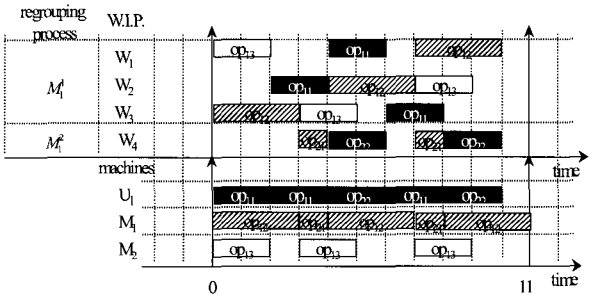
$\varphi(m_i)$ 는 기계 i의 총 작업 시간

$\gamma(m_i)$ 는 기계 i에 있는 토큰의 수, 즉, 트랜지션의 자원 분배를 말한다.

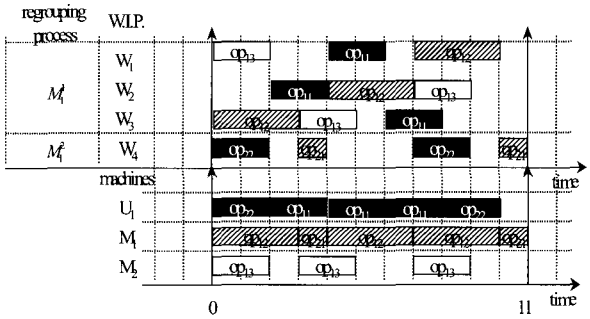
이러한 Unfolding 알고리즘에 그림 1의 스케쥴을 구하면 그림 11과 같다.



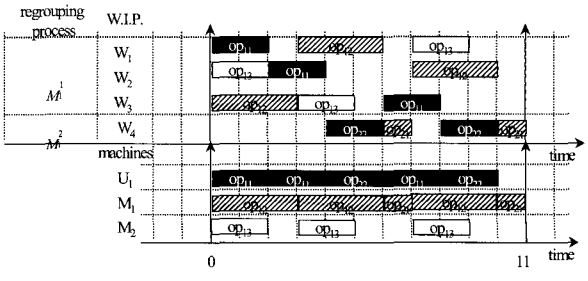
(a) Unfolding 알고리즘의 결과 1.



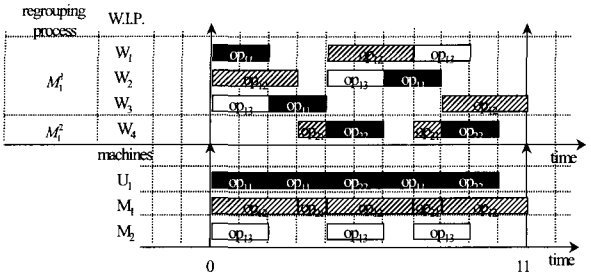
(b) Unfolding 알고리즘의 결과 2.



(c) Unfolding 알고리즘의 결과 3.



(d) Unfolding 알고리즘의 결과 4.



(e) Unfolding 알고리즘의 결과 5.

그림 11. Unfolding 알고리즘의 결과.

Fig. 11. The Result of the Unfolding Algorithm.

표 6. WIP의 수 비교.

Table 6. Compare of the WIP number.

	Hillion 알고리즘	Korbaa 알고리즘	Unfolding 알고리즘	분할 스케쥴링 알고리즘
WIP의 수	6	3	4	4

4. 효율성 분석

이 절에서는 패트리넷을 기초로 한 분석 알고리즘들을 비교 분석한 [12]를 근거로 하여, WIP의 수와 알고리즘 복잡도와 대기손실시간을 분석 요소로 하여, 분할 스케쥴링 알고리즘의 효율성을 분석한다.

4.1. WIP의 수

Hillion의 스케쥴은 6개의 작업 운반대, Korbaa는 3개, Unfolding은 4개의 작업 운반대와 5개의 결과를 얻었다. 그리고 분할 스케쥴링 알고리즘은 4개의 작업대와 2개의 결과를 얻었다. 이러한 결과들에서 우리는 Korbaa의 스케쥴이 가장 좋고, Unfolding의 스케쥴과 분할 스케쥴링 알고리즘의 스케쥴이 Hillion의 스케쥴보다 좋다는 것을 알 수 있다. 이를 표로 나타내면 표 6과 같다.

4.2 계산과정수 복잡도

Hillion 알고리즘은 각 프로세스의 비교시간이 120 이고, 두번째 단계와 마지막 단계를 합쳐 총 126번의 비교가 있게 되므로 120*126=15120 의 계산과정상의 복잡도가 구해진다. Korbaa 알고리즘은 부분화 9번, 이에 따른 각각의 재그룹화 34번, 계산사이클 52번이므로 이를 모두 계산과정에 포함하면 9*34*52=15912 이다. Unfolding 알고리즘은 각 BUC마다 모든 가능한 순열을 포함하여 계산하므로 본 연구의 예제모델에 대한 계산과정상의 복잡도는 240+120*4+4*6=744 이다. 한편, 본 연구에서 제안한 분할 스케쥴링 알고리즘은 분할 3, 각 분할된 서브넷의 테이블 계산 13번으로 총 16번의 계산과정과 각 기계별 (U1, M1, M2) 스케쥴 계산 3번을 조합의 수로 곱한 값만큼의 계산 과정이 더 필요하므로 총 16+(3*조합의 수)의 계산이 필요하다. 그러므로 여기서는 16+(3*2)=22번의 계산과정이 있게 된다.

이러한 계산과정상의 복잡도를 표로 정리하면 표 7과 같다.

4.3 대기 손실 시간

Hillion 알고리즘은 W1에서 4만큼의 대기 손실 시간이 필요하고, W2에서는 4, W3은 4, W4는 8, W5는 9, W6은 10 만큼의 대기 손실 시간이 있으며, 이러한 대기 손실 시간이 길면 길수록

표 7. 계산과정수 복잡도 비교.

Table 7. Complexity Compare of the Computation Process.

	Hillion 알고리즘	Korbaa 알고리즘	Unfolding 알고리즘	분할 스케쥴링 알고리즘
계산 과정 수의 비교	15120	15912	744	22

표 8. 대기손실시간 비교.

Table 8. The Waiting Loss Time Compare Table.

	Hillion 알고리즘	Korbaa 알고리즘	Unfolding 알고리즘	분할 스케쥴링 알고리즘
대기손실시간 (Waiting Loss Time)	39	6	17	17

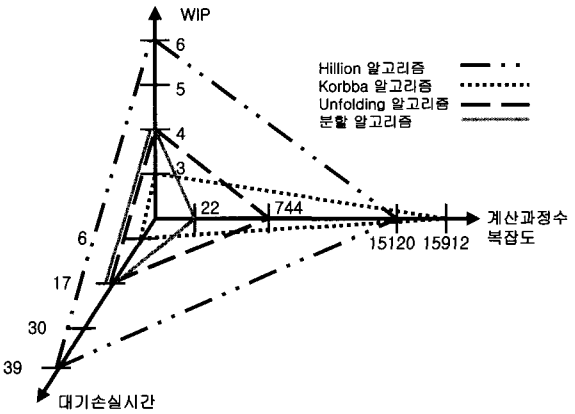


그림 12. 알고리즘 간의 관계 그래프.

Fig. 12. The Relation Graph of Algorithms.

그 알고리즘은 효율적이지 않다고 할 수 있다. Korbaa알고리즘은 W1에서 1, W2에서의 대기 손실 시간은 없으며, W3에서는 5만큼의 대기 손실 시간이 필요하다. Unfolding 알고리즘은 W1, W2, W3에서 각각 4만큼의 대기 손실 시간이 있게 되고, W4에서 5만큼의 대기 손실 시간이 걸린다.

한편, 분할 스케쥴링 알고리즘 또한 W1, W2, W3에서 각각 4만큼, W4에서 5만큼의 대기 손실 시간이 걸리게 된다. 이러한 대기 손실 시간 측면에서의 효율성 비교를 하면 다음 표8과 같은 비교표를 얻게 된다.

이러한 결과들을 토대로 각 알고리즘간의 관계 그래프를 도식하면 그림12와 같다.

VI. 결론

이 연구에서, 분할 스케쥴링 알고리즘과 기존의 스케쥴링 알고리즘들과 종합적으로 비교 분석 하였을 때, WIP의 수 면에서는 Hillion알고리즘 보다는 양호한 결과를 얻을 수 있었으나, Korbaa 알고리즘보다는 좋은 결과를 얻지 못했으며, Unfolding 알고리즘과는 같은 결과를 얻을 수 있었다. 그러나 알고리즘 계산과정 상의 복잡도 측면에서의 효율성 면에서는 계산 과정이 단순 함으로 해서 Korbaa나 Hillion 알고리즘

보다는 좋은 결과를 얻을 수 있었다.

즉, WIP만을 가지고 분석하면 Korbaa의 알고리즘이 우수하고, WIP뿐만이 아니라 계산이나 시간상의 경제성을 고려한다면 분할 알고리즘이 더 효율적이다.

이 연구에서, 모델의 분할을 기본으로 분석하고자 한 것은 분석할 경우에 발생 할 수 있는 상태 폭발을 최소화 하고자 함이었다. 또한 기존의 모듈화 패트리넷 연구가 합성, 수학적 그리고 도달성 분석보다는 효율적이기는 하나, 작성자의 임의의 기준에 의하여 시스템이 분할되어 다양한 모델링 편의와 작성이 어렵고, 시스템마다 다양한 기준들을 적용해야 하는 반면, 분할 알고리즘은 어떠한 시스템 모델링에 적용하더라도 동일한 방법과 기준을 적용하여 분할, 분석 함으로써 이에 따르는 효율성을 증가시킬 수 있었다.

참고문헌

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

[2] 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.

[3] 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.

[4] H.Camus, Conduite de systèmes Flexibles de Production Manu- facturière par composition de régimes permanents cycliques : modélisation et évaluation de performances à l'aide des Réseaux de Petri, Thèse de doctorat USTL 1,mars 1997.

[5] 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.

[6] J. K. Lee, Une méthode d'analyse d'ordinnancement des systems fléxibles de production manufacturière utilisant le dépliage des réseaux de Petri, Thèse de doctorat EC Paris, mars 2002.

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

[8] 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, Thèse de doctorat USTL 1,juillet 1998.

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

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

[11] 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.

[12] J. K. Lee, Benchmarking Study of the Cycle Scheduling Analysis Methods in FMS, *In : proceeding IEEE SMC'02,* 2002.

송 유 진

1969년 6월 10일생. 1992년 창원대학교 컴퓨터공학과 졸업(학사). 1995년 창원대학교 대학원 컴퓨터공학과 졸업(석사). 2003년 창원대학교 대학원 컴퓨터공학과 (공학박사) 1995년~2002년 창원대학교 강사. 2003년~현재 창원대학교 초빙교수. 관심분야: 패트리넷, 성능분석, 정보보호, 스케쥴링 연구.

이 종 근

1952년 4월 28일생. 1974년 숭실대학교 전자계산학과 및 동대학원 공학 석사. 1978년 고려대학교 경영대학원 경영학 석사. 1987년~1990년 LSI / Univ. de Montpellier II 연구원 역임. 2002년 LCGI / Ecole Centrale Paris 전산학, 공학박사. 1983~현재, 창원대학교 컴퓨터공학과 교수. 컴퓨터공학과장, 전산소장,자연대부학장역임. 관심분야 : 패트리넷, 스케쥴링 분석, 성능분석, 정보보호 관련 연구.