Journal of the Korean Institute
of Industrial Engineers
Vol.21, No.2, June. 1995

분산관리 시스템을 위한 동적 스케쥴링의 연구*

A Study on the Dynamic Scheduling for Distributed Management Systems

정 남 기**

Namkee Chung

Abstract

Constructing a distributed management system has its own advantages in addressing the issue of implementing a quick responsive management system in dynamically changing environment of enterprise. We suggest a basic scheduling methodology applicable to a distributed production management system. A new concept of "flexible schedule" is introduced as a tool to accommodate dynamically changing situations of job shops. Then a search technique (referred to as CSP-CBA search) is presented to obtain such a schedule for the job shop scheduling problem, which is converted into a constraint satisfaction problem(CSP), by using the constraint based analysis(CBA). This approach is tested on more than 100 test problems. The results show that the suggested approach required shorter CPU time and solved more problems in given time than another fixed schedule method.

1. 서론

분산관리 시스템은 여러개의 일잡이(agent)들이 총과업(global task)을 하위과업(subtask)으로 분할하여 수행하는 시스템으로, 중앙집중 시스템에 비해 모형화, 병행처리, 모듈화, 신뢰성에서 잇점을 가지고 있다. 하나의 일잡이는 소프트웨어를 의미하며, 각각 독립적으로 일을 처리하되 필요에 따라 서로 정보를 교환한다.

최근에는 분산관리 방식이 기업 활동 여건 변화를 적시에 수용하기 위한 방법으로 도입

* 이 연구는 1993년도 교육부 국비 해외파견 연구 지원에 의해 수행되었음.

** 전남대학교 산업공학과

되고 있다. 제조기업에서 원료공급/제조, 제조/분배, 그리고 분배/소비 간의 관계를 "공급사슬(supply chain)"로 파악하고, 이 공급사슬을 통합적으로 관리하기 위하여 분산관리 방식을 채용한다.

여기서 먼저 생각되는 것은, 요구되는 각 관리활동들을 어떠한 일잡이에 어떻게 담당시킬 것인가 하는 문제이다. 일반적으로는, 각 일잡이 사이에 가능한 많은 정보가 공유되고 교환될 수 있도록 일잡이를 정의하고 담당 관리활동을 정해준다. 예를 들어, Fox[3]는 다(多)공장 제조기업의 공급사슬 분산관리를 위하여 7개의 일잡이, 즉 주문(order acquisition), 자원(resource management), 물류(logistics), 수송(transportation), 스케쥴링, 작업분배(dispatching) 그리고 정보(information) 일잡이들을 정의하였다.

Fox는 여기서 스케쥴링 일잡이의 역할을 중요시한다. 이것은 MPS(master production scheduling)와 상세 스케쥴링의 두 기능을 함께 처리할 수 있도록 정의한다. 물류 일잡이로부터 건네지는 생산요구 정보, 자원 일잡이로부터 오는 자원 현황, 그리고 작업분배 일잡이로부터 현재 스케쥴을 입력받아, 생산계획/스케쥴을 정한다.

분산관리 방식에서 스케쥴링 일잡이는 동적(動的) 공장상황을 반영할 수 있어야 한다. 기계고장, 원자재.부품의 납기 지연, 불량발생과 재작업, 작업자의 결근, 등 예기치 못한 갑작스런 상황이 발생할때, 스케쥴링 일잡이는, 이 변화들을 새로운 제약조건으로 삼고, 이 제약들을 만족시키는 새로운 스케쥴을 생성시킬 수 있어야 한다. 이런 상황 변화가 감안되지 않는 정적(靜的) 스케쥴링은, 공장의 운용 부문보다는 계획 부문에 제한적으로 사용될 수밖에 없으므로, 분산관리시스템의 구축에 부적절하다.

이 연구는, 분산관리 시스템에서 채택될 수 있는 동적 스케쥴링 방법으로 제약만족(constraint satisfaction) 스케쥴링을 제안한다. 납기, 공정 재고수준, 기계교체 횟수 등, 공장경영 목표를 제약조건으로 표현하고, 이 조건들을 만족시키는 스케쥴을 찾는 방법이다. 분산관리 시스템에서 제약만족 스케쥴링이 적합한 것은, 다른 일잡이들이 스케쥴링에 요구하는 조건들을 제약으로 쉽게 표현할 수 있기 때문이다. 이 연구는 특별히 납기만족을 중시하여, 납기 제약만족 스케쥴링이 공장상황의 동적 변화에 대처할 수 있음을 보이려 한다. 이 연구의 결과는 동적 스케쥴링 시스템의 기반이 될 수 있으리라 기대한다.

제약만족 스케쥴링 방법은 디스패칭(dispatching) 방식의 스케쥴링과는 다르다. 디스패칭 방식은 납기만족을 보장할 수 없으며, 단지 스케쥴을 구한 후 납기 만족여부가 평가된다. 따라서 디스패칭 방식의 스케쥴은, 다른 일잡이들의 요구사항을 만족시킨다는 보장이 없으므로 분산관리 시스템의 스케쥴 기능을 충분히 수행하기 곤란하다.

이 방법은 인공지능 기법의 일종인 CSP (constraint satisfaction problem) 모형을 활용한다. 먼저, CSP 모형을 활용한 제약만족 스케쥴링에 관한 연구 현황을 살펴보자.

ISIS[4]는 CSP 모형을 도입한 최초의 스케쥴링 시스템이다. 이것은 터빈 공장을 모델로 하여, 5가지 형태의 제약조건들, 즉, 경영목표, 물리적 제약, 인과(因果) 제약, 활용가

능 상태에 대한 제약, 그리고 선호 항목 등을 고려하고 있다. OPIS[8]는 ISIS의 단점을 보완하기 위해 개발된 시스템이다. 이것은 자원, 주문, 사건에 따라 문제형태를 달리 설정한다. 그런데, 이 시스템들은 계산시간이 길어 예기치 못한 돌발적 사건에 대처하는 능력이 부족하다.

일반적인 Job Shop에서 제약만족 스케줄링 문제에 CSP 모형을 적용 하려는 연구도 활발하다. 이들의 목표는 보다 크고 복잡한 규모의 문제에서, 제약을 만족하는 가능 스케줄을 보다 빠른 시간에 찾으려는 것이다. 이들을 부분 스케줄(partial schedule)을 생성하는 방법에 따라 분류하면, 공정순서를 지정하는 순서지정방법[10]과 공정 시작시간을 지정하는 시간지정방법[1,5,7,9]으로 나뉜다. 예를 들어, 하나의 기계를 두고 두 공정이 서로 경합할때, 전자는 공정의 기계사용 순서를 정해줌으로써 스케줄을 생성해가는 방법이고, 후자는 공정별 작업시작 시간을 정해가면서 최종적으로 하나의 스케줄을 찾아내는 것이다. Sadeh[9]의 MicroBOSS는 시간지정방법에 의한 Job Shop 스케줄링 시스템으로, 작업시작시간이 지정될 때마다 각 기계의 부하(負荷)를 새로 추정하여 다음 시간지정에 활용하고 있다. Muscettola[6]의 연구는 기본적으로 시간지정방법을 사용하나, 탐색의 능률을 높이기 위해 순서지정의 아이디어를 활용하였다.

이 논문은, Job Shop에서 납기를 만족하는 융통성있는 스케줄을 얻고자 한다. 크고 복잡한 규모의 문제를 짧은 시간에 해결할 수 있는 방법을 제시하려 하며, 이를 위해 순서지정방법을 사용하려한다. 2절에서 융통성있는 스케줄의 의미를 더 상세히 설명하고 3절에서는 CSP 모형에서 순서지정방법 사용을 설명한다. 4절에서 구체적인 스케줄 탐색과정을 제시한 후, 5절에서 이에 대한 계산 실험 결과를 MicroBOSS와 비교한다.

2. 분산관리 방식을 위한 융통성있는 스케줄링

납기만족을 확실하게 보장하는 스케줄은 분산관리시스템에 유용하다는 것을 설명하였다. 그런데, 납기만족 스케줄은 여러개가 가능하다. 이 스케줄들을 하나씩 파악하는 것보다, 이들의 집합, 즉 납기 만족을 보장하는 어떤 시구간(時區間)을 파악하여, 이 범위 안에 있는 스케줄을 한꺼번에 아는 것이 유리하다. 가능 스케줄 집합이 있으면, 스케줄을 고정시키지 않고, 상황의 변동에 따라 그중 적합한 하나를 선택할 수 있기 때문이다. 따라서 가능 스케줄 집합은 "융통성있는 스케줄"로도 해석된다. 납기를 만족시키는 범주에서 스케줄이 융통성있게 조절될 수 있다.

융통성있는 스케줄의 의미를 더 밝히기 위해 다음 그림 1을 보자. 이것은 한 기계에서 수행되는 4개 공정에 대한 스케줄이며, (i,j)는 주문 i의 j번째 공정을 나타낸다. 그림 1(a)는 각 공정의 시작 시각과 완료 시각(이를 작업시간대라 하자)이 고정되어 있어, 만약 어느 한 공정의 작업시간대에 변동이 있을 때, 그 변동이 전체 스케줄에 어떤 영향을 미치는지 가늠하기 어렵다. 그 작업시간대를 뒤로 옮겨도 납기는 만족되는지, 그렇다면 얼마 기간의 한도 내에서 옮길 수 있는지 전혀 알 수 없다. 이것은 단 하나의 가능

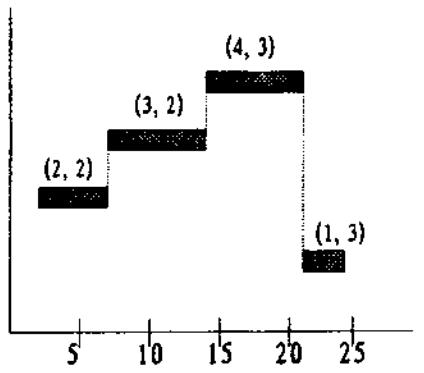
스케줄을 보여줄 뿐이며, 이것 이외의 다른 가능 스케줄을 찾기 위해서는 별도의 스케줄링이 필요하다.

그러나, 그림 1(b)는 융통성있는 스케줄로서, 가능 스케줄 집합을 보여준다. 각 작업시간대에 곁들여져 있는 가는 실선이 가능 작업시간대를 나타낸다. 즉, (2,2)가 1분 늦게 시작되어도, 그 뒤따르는 공정들이 납기를 어긋나지 않게 조정될 수 있고, (3,2)가 2분 늦게 시작되어도 지장없음을 보여주고 있다.

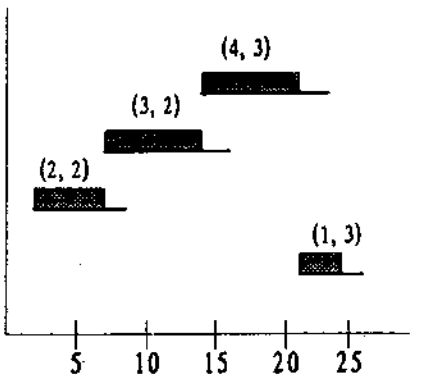
유있게 설정하거나, 가공시간에 여유시간을 더하거나, 대안기계를 마련하면 융통성이 커지고, 재공재고를 감소시키기 위해서는 융통성을 작게 해야한다.

3. CSP 모형의 활용

이처럼 융통성있는 스케줄을 구하려할 때, CSP 모형은 유용하다. 사실 Job Shop의 스케줄링 문제는 CSP 모형이 잘 활용될 수 있는



(a) 융통성 없는 스케줄



(b) 융통성 있는 스케줄

그림 1. 스케줄 융통성의 비교

기계 고장이나 부품조달의 어려움 등, 예기치 않은 사정을 감안할 때, 정확하게 명시된 스케줄보다 융통성 있는 스케줄이 더 유리하다. 이 연구는, 납기를 만족시키면서도 작업분배 일잡이에게 융통성을 제공해주는 동적 스케줄링 방법을 제시한다. 납기를 만족시키는 융통성있는 시구간을 정하여 둠으로써, 예기치 않은 상황 변화에 대처할 수 있게 된다.

이 융통성의 크기는 다른 일잡이와의 관계를 반영하여 설정된다. 예를 들어, 납기를 여

대표적인 문제이다. 설명을 돕기 위해 다음의 5개의 주문과 3대의 기계에 대한 Job Shop 스케줄링 문제에 대한 CSP 모형을 생각해 보자.

주문	도착시각	납기	공정별 사용기계	공정별 가공시간
1	0	26	2 3 1	3 6 3
2	0	26	2 1 2 3	2 5 2 7
3	0	26	3 1 2	5 7 3
4	0	26	2 3 1 2	4 6 7 4
5	0	26	3 2	2 6

그림 2는 이 예제의 제약 그래프(constraint

graph)로, 노드는 각 주문의 공정을 나타내며, 아크의 화살 실선은 공정 선후 제약을, 화살없는 다른 선들은 기계사용 제약(한 기계에서 동시에 작업할 수 있는 공정은 하나라는 제약)을 나타낸다. 각 노드에 표시된 시구간은, 공정의 선후관계만을 감안할 때 납기를 만족시킬 수 있는 작업 가능시작시각(earliest start time)과 허용완료시각(latest finish time)을 표시한다.

로 간주하여, (1)은 변수 매김(variable ordering), (2)는 값 매김(value ordering)으로 해석된다.

Job Shop의 스케줄링 문제를 CSP 모형으로 해결하려는 연구는, 문제의 특성(주로 자원의 용량에 대한 제약)을 활용하여, 제약전파(constraint propagation) 방법, backtracking 방법, 공정의 선택(변수 매김)과 작업시간대의 할당(값 매김) 방법을 찾는데 노력해 왔

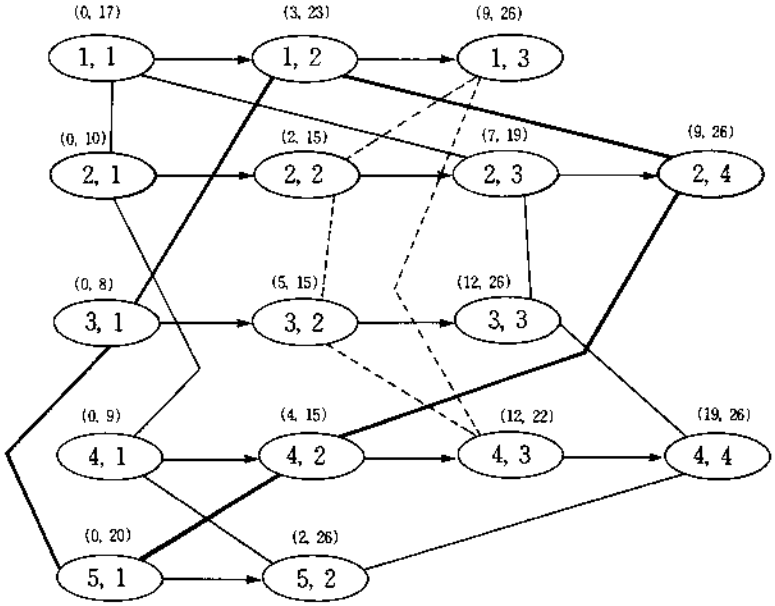


그림 2. 예제에 대한 제약 그래프

그런데, 한 기계를 복수공정이 중복사용하지 않는다는 기계사용 제약까지 만족시키기 위해서는, 이 시구간들이 더 조정되어야 한다. 기계사용 제약을 만족시키려 할때, (1) 어느 공정(노드)을 먼저 선택할 것인가? (2) 그 공정의 작업시간대로 어떤 값을 할당할 것인가? 의 2가지 문제에 부딪힌다. CSP 모형의 관점에서, 공정을 변수로, 작업시간대를 값으

다[1,5,7,9,10]. 그런데, 지금까지 주로 사용되었던 시간지정방법은 한 공정당 한개씩의 작업시간대를 할당해가면서 최종적으로 하나의 스케줄을 찾아내는 것이므로, 분산관리 방식을 위한 융통성있는 스케줄을 생성하지 못한다. 뿐만아니라, 이들은 탐색과정에서 계산 부담이 크다는 문제점이 있다.

이 연구는 이를 해결하기 위한 방법으로,

동일한 기계를 사용하는 공정들간의 공정순서를 정해주는 순서지정방법을 사용하려 한다. 하나의 기계를 2개의 공정이 같은 시간에 사용하고자 경합할 때, 이 공정들 간의 공정 순서만 조정하면 기계사용제약은 해결 될 수 있다. 이런 취지에서 다음과 같은 제약기반분석(CBA : Constraint-Based Analysis)을 사용한다[2].

공정 i, j의 가공시간을 각각 p_i, p_j, 그리고, 현재 시구간을 각각 (s_i, f_i), (s_j, f_j)라 하자. $\lambda = f_i - s_j$, $\mu = f_j - s_i$, 그리고, $\sigma = p_i + p_j$ 를 계산해 둔다.

(경우 1) 한공정이 다른 공정보다 먼저 작업되어야 한다. 즉, $f_i - s_j < p_i + p_j < f_j - s_i$ 일때, 공정 i가 공정 j를 앞서야한다.

(경우 2) 두 공정간 어떤 순서도 허용된다. $p_i + p_j > f_i - s_j$이며 $p_i + p_j > f_j - s_i$ 일 때 이다.

(경우 3) 현재 시구간에서는 가능 스케쥴이 없다. $p_i + p_j < f_i - s_j$이며 $p_i + p_j < f_j - s_i$ 일 때 이다.

이 분석은 납기만족 스케쥴을 보장하면서 기계사용제약을 해결하는 데 도움이 된다. (경우 1)일때, 새로운 공정 선후관계 제약이 추가되며, 이 영향으로 시구간이 좁혀지고 두 공정간 기계사용제약은 해결된다. 만약, 같은 기계를 사용하는 모든 공정간의 관계가 전부 (경우 1)로 나온다면, 결과적으로 나타나는 시구간들이 그대로 융통성있는 스케쥴이 된다.

한편으로, (경우 2)일때는, 스케쥴을 생성하기 위해서는 이들간 공정 우선 순위가 어떤 방식으로든 정해져야 한다. (경우 3)은, (경우 2)에서 우선순위가 잘못 정해졌거나, 납기가 너무 빨라 가능 스케쥴이 없는 경우이다. 이 두 경우에 대처하는 방법은 다음 절에서 다루어진다.

4. 스케쥴 탐색 과정

이 절에서는 3절에서 설명된 CSP 모형과 CBA를 이용하여, 융통성있는 가능 스케쥴을 찾아가는 과정을 설명한다. 다시말하면, 이것은 CBA에서 결정될 수 없는 공정 순서 (경우 2)를 인위적으로 정해가는, 스케쥴 탐색 과정이다.

이 연구에서, 제약전파와 backtracking이 교대로 사용되는 나무 탐색(tree search) 방법이 사용된다. 먼저, CBA를 적용하여, 어느 순서나 허용되는 (경우 2)가 나타나면, 인위적으로 그들간의 순서를 하나하나씩 정한다. 이 때마다 CBA는 제약만족을 검사(공정 순서와 가능해 존재 여부를 확인)한다. 그 결과 추가되는 제약이 있으면(경우 1), 이를 전파하여 시구간을 좁힌다. 만약 CBA에 의해 지금까지 추가된 제약들에 의해 가능해가 없다고 판정되면(경우 3), backtracking하여 종전의 결정을 조정한 후 다시 탐색한다. Smith와 Cheng[10]의 연구도 CBA를 활용하나, backtracking 없는 탐색에 의존하므로 탐색의 범위가 제한되고 있다.

우리는 이 스케쥴 탐색 과정을 "CSP-CBA"로 표기하기로 하며, 그 흐름을 구체적으로 다음과 같이 정리한다.

스케쥴 탐색 과정 CSP-CBA

```
do CBA 적용
while (경우 2) 존재
    변수 매김
    값 매김
    CBA에 의한 제약만족 검사
  if (경우 3)
    backtracking
  endif
endwhile
```

CSP-CBA에서 변수 매김과 값 매김은 탐색의 횟수를 줄이는 데 역할이 크다. 변수는 같은 기계를 사용하는 한 쌍의 공정을 나타내고, 값은 이 쌍간에 공정 선후 관계를 의미한다. 예를 들어 공정 a와 b가 같은 기계를 사용하려 할때, 변수 x_{ab}는 a보다 b를 먼저 작업시킬 때 0, 그 반대일 때 1의 값을 갖는다. 변수 매김 방법으로, 일반적인 CSP 모형에 대한 실험 결과와 스케쥴링 문제의 특수한 구조를 감안하여 다음 4가지를 선정하였다.

변수 매김 방법

(제약) 인접한 제약이 많은 것.
인접한 아크가 많은 노드를 우선적으로 선택하는 것이 좋다는 CSP 모형에 대한 연구 결과를 활용하였다.

(공정) 후행 공정 수가 많은 것.
디스패칭에서 흔히 사용되는 경험적 규칙이다.

(여유) 여유시간이 작은 것[10].
공정의 작업시간에 비해 시구간이 좁아 여유시간이 작은 것을 먼저 택한다.

(임의) 임의적인 선택.
위 3가지 기준과 비교하기 위함이다.

값 매김 방법으로는, 최소지연시간(最小遲延時間)이라는 개념을 설정하고, 이를 다음과 같이 계산하였다. 3절에서 정의된 부호를 사용할 때, 한 쌍의 공정 i, j의 지연시간 δ_i, δ_j 는,

$$\delta_i = s_i + p_i - s_j,$$
$$\delta_j = s_j + p_j - s_i.$$

여기서 δ_i는 i를 j보다 선행시킬 때 j가 자신의 시작가능시각보다 늦어지는 시간으로, 그 탓을 i에 돌릴 수 있는 기간이다.

값 매김 방법

(최소지연시간) $\delta_i < \delta_j$ 일 때, i를 j보다 선행시키고, 그렇지 않으면, 반대 순서를 적용한다.

CSP-CBA는 계산의 부담이 적다. CBA에 의한 제약만족 검사는 계산이 간단하고, 제약의 전파도 단순히 공정의 선후관계만을 추가하여 시구간을 다시 계산하는 것이다.

5. 실험 결과 분석

CBA를 이용함으로써, 융통성있는 스케쥴을 얻을 수 있는 잇점은 이미 설명되었으나, 어떠한 종류의 문제를 해결할 수 있으며, 얼마나 많은 시간이 걸리는지에 대해서는 실험을 통하여 확인하였다.

CSP-CBA는 4절에서 설명된 탐색과정과 변수 매김, 값 매김 방법으로 수행되었으며, 단순 backtracking에 의해 탐색되었다. 비교를 위하여 시간지정방법으로써 널리 알려진 MicroBOSS를 함께 설치하여 같은 문제를 풀도록 하였다.

이 실험은 C++ 언어를 사용하여 DECStation 5000/25에서 수행되었다. Sadeh[9]가 만든 60개의 문제와 Nuijten이 수집한 문헌상의 연구용 문제 51개를 풀어, 풀린 문제의 수와 푸는데 걸린 CPU 시간을 비교하였다.

〈표 1〉은 Sadeh가 만든 60개의 문제로 실험한 결과이다. 각 문제의 크기는 10×5(주문×기계)이고, 문제 유형은 애로공정(bottleneck) 2수준, 납기의 다급함(looseness) 3수준을 생각하여 6가지이며, 유형별 10 문제씩이다. 변수 매김에 따른 CSP-CBA의 4가지 방법과 Sadeh의 MicroBOSS 방법을 비교하였다. 그 결과, CSP-CBA 탐색 방법이 MicroBOSS에 비해 문제 해결 능력과 계산 시간에서 더 나은 결과를 보인다.

표 2는 더 다양한 문제에 대한 결과를 보여준다. 여기 사용된 51개의 문제들은 문제의 크기와 특성이 각각 다르다. 여기서도 CSP-CBA 가 MicroBOSS보다 더 나은 결과를 보인다.

CSP-CBA에서 값 매김 4가지 방법중, (임의) 방법이 풀린 문제 수에서 약간 적으나, 전반적으로는 그들간 우열을 가리기 힘든 정도로 큰 차이는 없다고 보여진다. 또, Sadeh 문제중 2 문제만이 4가지 방법중 어느 것으로도 (정해진 탐색시간 동안) 풀리지 않았다. 이는 값 매김 방법을 고정시켜 어느 한 방법만을 사용하는 것보다, 가능한한 여러 방법들을 병행적으로 사용하는 것이 바람직하다는 것을 보여주며, MicroBOSS에 비해 문제 해결능력이 크다는 것을 보여주는 점이다. 이 점은 CSP-CBA가 계산시간이 짧으므로 현실적 응용성이 있다고 생각된다.

6. 결론

분산관리로 구현되는 생산관리 시스템에서, 스케줄링은 다른 일잡이들의 활동에 가장 많은 영향을 끼치며, 공급사슬 관리의 성패에 직접적인 영향을 준다. 스케줄링 일잡

표 1. Sadeh 문제에 의한 CSP-CBA와 Micro-BOSS의 탐색 능력 비교

(제한시간 600 CPU sec)

문제			풀린 문제 수					소요시간				
유형	크기	갯수	CSP-CBA				MicroBOSS	CSP-CBA				MicroBOSS
			제약	공정	여유	임의		제약	공정	여유	임의	
1		10	5	9	3	6	5	16	22	12	14	123
2		10	5	9	7	4	6	21	39	32	17	29
3	10	10	10	10	9	9	9	23	17	10	13	124
4	x	10	9	10	10	9	8	52	16	58	30	121
5	5	10	10	10	10	9	7	13	17	7	13	133
6		10	9	10	10	9	9	14	16	14	16	98

표 2. Nuijten 수집 문제에 의한 CSP-CBA와 MicroBOSS의 탐색 능력 비교

(제한시간 18000 CPU sec)

문 제			풀린 문제 수					소 요 시 간				
出處[a]	크기	갯수	CSP-CBA				Micro BOSS	CSP-CBA				Micro BOSS
			제약	공정	여유	임의		제약	공정	여유	임의	
ft	6× 6	1	1	1	1	1	1	2	3	1	2	5
	10×10	1	1	1	1	1	1	83	93	67	89	1593
	20× 5	1	0	0	1	0	1	-	-	289	-	929
car	11× 5	1	1	1	0	1	0	32	28	-	32	-
	13× 4	1	1	1	0	1	1	30	32	-	37	5961
	12× 5	1	1	1	0	1	0	44	50	-	36	-
	14× 4	1	1	1	0	1	1	52	61	-	50	17730
	10× 6	1	1	1	0	1	0	28	17	-	47	-
	8× 9	1	1	1	1	1	0	34	36	1638	28	-
	7× 7	1	1	1	1	1	0	8	7	7	9	-
	8× 8	1	0	1	1	1	0	-	15	1508	24	-
la	10× 5	5	5	5	5	5	5	15	15	7	15	316
	15× 5	5	5	5	5	5	4	95	120	48	101	1286
	20× 5	5	3	1	5	2	3	511	2608	385	503	6338
	10×10	5	5	5	5	4	3	90	122	46	96	1230
	15×10	5	5	5	5	4	0	885	1322	802	763	-
orb	10×10	10	10	10	10	8	8	95	123	49	85	1283
abz	10×10	2	2	2	2	2	2	68	129	102	104	1433
	20×15	3	3	2	3	0	0	13177	14948	4876	-	-

[a]ft : Fisher and Thompson in "Industrial Scheduling" edited by Muth and Thompson(1963).

car : "Ordonnancements a constraints disjunctives" by J. Carlier(1975).

la : "Resource constrained project scheduling : an experimental investigation of heuristic scheduling techniques" by S. Lawrence (1984).

orb : generated in Bonn in 1986.

abz : "The shfting bottleneck procedure for job shop scheduling" by J. Adams, E. Balas and D. Zawack(1988).

이의 역할은 다른 일잡이들로 구성된 "오케스트라"의 연주(활동)를 지휘(조정)하는 "지휘자"와 같다고 볼 수 있다.

이 연구는, 분산관리 방식에서 응용될 수 있는 동적 스케줄링에 대한 접근 방법으로, "융통성있는 스케줄"의 개념을 도입하였다. 납기만족을 목표로 삼는 Job Shop 스케줄링 문제에서 이를 CSP로 모형화하고, 여기서 "융통성있는 스케줄"을 탐색하기 위해 CBA를 사용하였다. 이 CSP-CBA 모형에서 변수 매김 방법을 4가지 사용하고, 최소지연시간을 기준으로 값 매김하여 단순 backtracking에 의해 스케줄을 탐색하였다. Sadeh의 문제 60개와 Nuijten이 수집한 51개의 각종 문제를

대상으로 CSP-CBA를 실험한 결과, Sadeh의 MicroBOSS 방법보다 더 많은 문제를 보다 빠른 시간에 풀 수 있었다.

여기서는 단순 backtracking을 사용하는데 그쳤으나, 만일 불가해(不可解)를 야기시킨 결정으로 되돌아가는 지능적 backtracking이 적용된다면 더 나은 결과를 기대할 수 있을 것이다. 이것은 더 큰 규모의 문제를 해결하려 할 때 필요성이 더 커진다. 앞으로 더 연구되어야할 또 다른 과제로서, 기계(자원)에 관계되는 여러 제약조건들, 예를 들어, 치공구, 준비시간, 용량, 대안기계의 활용 등을 추가로 고려하여 모형화하는 것이 있다. 보다 현실에 더 가까운 이 모형에서는, 탐색하는 기법으로 CBA가 단순히 적용되기보다는 다른 기법과 함께 다른 각도에서 응용되는 것이 바람직할 것으로 생각된다.

참 고 문 헌

[1] Davis, E.D., ODO : A constraint-based scheduler founded on a unified problem solving model, Master's Thesis, Dept. of Comp. Sci., University of Toronto, Ontario, Canada, 1994.

[2] Erschler, J., Roubellat, F., and Vernhes, J.P., "Finding some essential characteristics of the feasible solutions for a scheduling problems," *Operations Research*, Vol. 24, 772-782, 1976.

[3] Fox, M.S., Chionglo, J.F, and Barbuceanu, M., The integrated supply management system, Working Paper, University of Toronto, Ontario, Canada, 1993.

[4] Fox, M.S., and Smith, S.F., "ISIS – A knowledge-based system for factory scheduling," *Expert Systems*, 1(1), 25-49, 1984.

[5] Keng, N. and Yun, D.Y.Y., "A planning /scheduling methodology for constrained resource problem," In *Proc. IJCAI-89*, Detroit, MI, 1989.

[6] Muscettola, N., "Scheduling by iterative partition of bottleneck conflicts," In *Proc. 9th IEEE Conference on AI Applications*, Orlando, FL, 1993.

[7] Nuijten, W.P.M., Aarts, E.H.L., van Erp Taalman Kip, D.A.A. and van Hee, K.M., "Randomized constraint satisfaction for job shop scheduling," In *Proc. IJCAI-93 Workshop on Knowledge-Based Production Planning, Scheduling, and Control*, Chambery, France, 1993.

[8] Ow, P.S., and Smith, S.F., Viewing scheduling as an opportunistic problem-solving process, Working Paper, Carnegie Mellon University, Pittsburgh, PA, 1986.

[9] Sadeh, N., Micro-opportunistic scheduling : The Micro-boss factory scheduler, The Robotics Institute, Carnegie Mellon Univ., Pittsburgh, PA, 1993.

[10] Smith, S., and Cheng, C.C., "Slack-based heuristics for constraint satisfaction scheduling," In *Proceedings of the 11th Natioal Conference on AI*, July 11-15, Washington DC, 1993.