객체에 근거한 선호도 제약 중심 스케줄링 언어와 성능향상 기법*

이기철** · 문정모*** · 송성헌****

An Object-Based Preference-Driven Scheduling Language and Techinques for Improving Its Perforance*

Kee-Cheol Lee** · Jung-Mo Moon*** · Sung-Hun Song****

ABSTRACT

For a complex scheduling system like time table construction, its optimal solution, if exists, is hard to obtain. In this paper, the scheduling environment is reasonably confined as where objects have their own events competing for better slots on boards, and objects have their own board slot preferences and belong to one or more classes of the society which globally constrains them.

Here, two phase method is suggested, where the first phase is human-like preference dirven and the second phase is for fine tuning by considering all the factors given. Designed and implemented in our system HI-SCHED are dynamic object switching, temporal-constraint-driven intelligent backtracking, case-based revisions, object-based approach, and so on. Some satisfaction degrees are also defined to measure the usefulness of our method. In addition, look-ahead dynamic object switching is considered, and additional global constraints are introduced and processed. A simple scheme is also used to verify the usefulness of the post processing scheme.

* 본 논문은 한국과학재단 핵심전문 연구비 지원(과제번호 931-0900-017-2)에 의한 것임.
** 홍익대학교 컴퓨터공학과
*** 전자통신연구소
**** 홍익대학교 산업공학과

I. 서 론

인간은 매일 여러가지의 계획을 세우고 이를 무리없이 수행해 간다. 이런 스케줄링의 기능은 기존의 학문에서 다루는 스케줄링 문제 해결의 접근 방법과는 많은 차이가 있음을 알 수 있다. 최근 인공지능 분야에서 모두가 공감할 수 있는 생산품으로 전문가 시스템을 들 수 있다. 과연 이와 같은 범용성을 갖는 범용 스케줄링 생성 도구의 생성은 가능할 것인가? 이러한 질문에 지금 답할 수는 없지만 우선 이런 도구가 가져야할 기본 성분은 무엇인가에 대한 질문을 던져 볼 가치가 있다.

우리가 일상적으로 대하는 스케줄링의 문제(가령 강의시간표 배정, 여행 스케줄 작성 등)의 최적해를 컴퓨터 프로그램으로 풀기가 쉽지 않다. 기술적으로 이야기할 때, 이런 문제는 NP-hard로 분류될 수 있는 부 문제(sub-problem)가 1 내지 수 개가 상호작용을 하는 문제 즉 복합 스케줄링 문제이기 때문에, 합리적인 시간에 컴퓨터에 의한 최적해를 기대할 수 없음은 당연하다. 이러한 복잡한 문제를 인간이 해결하고 큰 불편없이 살고 있는 이유를 분석해 보는 인지 과학적인 접근방법이 문제해결에 도움을 줄 수 있다. 기존의 접근 방식으로는 백트래킹을 이용한 방식이 있으나, 탐색공간의 방대함 때문에 문제가 되며, 중앙제어 방식 만으로는 스케줄링 문제의 복잡성을 다루는데 불충분하다. 우선순위를 갖는 결정 규칙에 의해 특정 항목들을 고려하는 접근방식이나, 단순모델에서의 최적해를 추구하는 방식은 실세계 문제풀이와는 거리가 있다.

결국 복합 스케줄링의 문제는 컴퓨터를 이용한 최적해를 합리적인 시간에 구할 수 없는 스케줄링 문제가 되는데, 이러한 복합 스케줄링 문제의 해결은 컴퓨터 기획, 스케줄링, 제약만족문제 및 객체지시형 프로그래밍과 직,간접으로 관련을 갖고 있다. 기획은 적용할 연산자(또는 스텝)의 순서를 계산하는 과정으로, 선형 기획[Fikes&Nilsson71], 계층기획[Sacerdoti74], 비선형기획[Sussman75][Sacerdoti75][Nilsson80] 등이 연구되었다. 80년대 들어, AI 연구가들은 스케줄링에도 관심을 갖게되어 비행기 상에서의 행위[Vere81], 기차스케줄생성[Ernesto93] [Fukumori80], 시간적 구조를 고려한 기획과정의 재편[Allen83] [Allen&Koomen83] [McDermott82], 전문 크리틱(critic)의 사용[Miller81][Stefik81a][Chapman87], 케이스에 근거한 기획[Hammond86], 작업장 스케줄링[Fox91] 등을 연구하였다. 제약만족문제(CSP)는 주어지거나 유도된 제약하에서 변수에 변수값을 배정하는 문제인데, 충고에 근거한 백트래킹[Dechter88]은 CSP로 복잡한 스케줄링형의 문제를 풀려는 시도이다. 프롤로그 III [Colmerauer90], CLP[Cohen90], CAL[Aiba88], 및 CHIP[Dincbas88] 등의 범용 제약언어는 CSP 를 해결하기 위한 범용 도구로 연구되었지만, 복잡한 스케줄링 시스템에는 적합하지 않다.

이상의 어느 방법도 복잡한 스케줄링 문제에 매우 적합하다고는 할 수 없으며, 이에 따라 본 연구에서 선호도에 근거한 방식[Lee94]의 언어인 HI-SCHED가 설계되고 구현되었다.

HI-SCHED 에서는 복합 스케줄링 기법의 특징의 최대 관건인 제약 만족 문제의 처리를 위해 OOP의 특성을 도입하여 객체 개념을 기반으로 문제를 해결하고자 하였다. 본 논문에서는 구현된 HI-SCHED 언어 자체보다는 HI-SCHED에서 사용한 인간방식의 선호도 제약에 근거한 스케줄링을 설명, 분석하고 그 개선방향을 제시하려 한다.

Ⅱ. HI-SCHED 의 소개

1. 스케줄링의 기본 요소와 충돌해결

1.1 스케줄링 환경과 용어

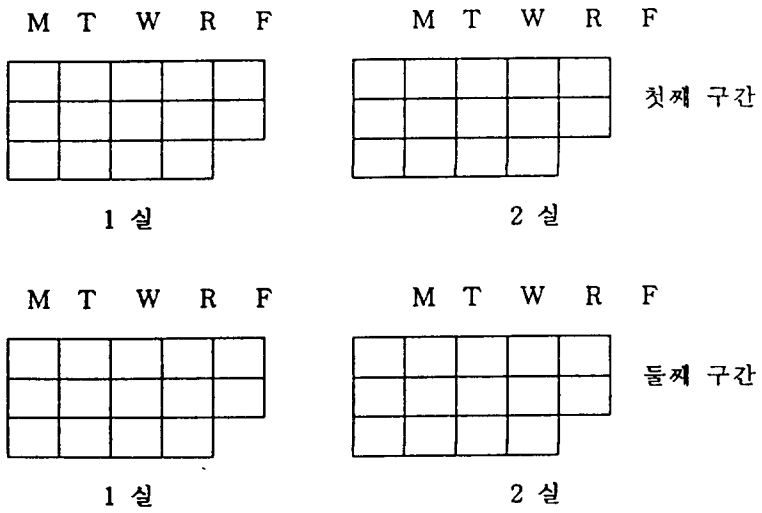
우선 스케줄링을 위한 환경을 정의하고자 한다. 다수의 스케줄링 문제와 부합될 만큼 일반성이 있으면서, 컴퓨터로 다룰 수 있을 만큼 접근 가능한 다음과 같은 환경을 가정한다.

스케줄링의 서비스를 원하는 고객을 객체(object)라 하며, 상호배제적인 공간 상의 단위 시간을 슬롯(slot)이라 하자. 상호배제적인 한 공간을 공유하며 서로 다른 단위시간을 갖는 슬롯들의 모임을 보드(board)라 한다. 각 객체는 배정을 원하는 사건(event)들을 가지는데, 각 사건은 사건명(event name)과 필요한 슬롯의 수(number of slots)로 구성되어 있다. 각 객체는 보드 상의 특정 슬롯이나 슬롯군에 대한 제약을 표시할 수 있는데, 그 제약은 반드시 원하거나 절대로 원하지 않는 제약적 제약(restrictive constraint)의 형태도 가능하고 선호적 제약(preference constraint)의 형태도 가능하다. 또한 사건 간에는 알렌의 시간추론[Allen83]에서 정의된 임의의 우선순위가 존재할 수도 있다. 객체는 자신을 전역적으로 제한하는 사회(society)의 1 개 또는 그 이상의 클래스(class)에 속한다. 이상의 간략화된 환경을 가령 단기 강좌 시간표 작성에 적용해 보면, 객체는 강사, 슬롯은 1 시간, 보드는 일주일 동안의 강의실에 해당되며, 선호도는 강사의 시간 슬롯에 대한 선호도를 간접적으로 표현한다. 또한 사건은 과목을 뜻하며, 선수과목 또는 동시 수강 가능과목 등을 과목간의 시간 우선순위로 표현할 수 있다. 객체를 전역적으로 제한하는 사회는 강좌개설 학교가 되며, 객체가 속한 클래스는 강사의 신분 등을 나타낼 수 있다. 객체를 전역적으로 제한한다는 것은 매우 여러가지를 뜻할 수 있는데, 가령 하루에 4 시간 이상을 강의할 수 없다든가 하는 문제이다. 여행 스케줄 작성 등의 예도 비슷한 매핑(maping)을 생각할 수 있다. 우리는 이런 기본 환경에서 공통적인 그리고 일반적인 모델을 정하고 범용성을 갖는 전역제약을 알아내어 내장하고, 클래스로 부터 상속받는 특성들과 다중상속의 해결 능력을 언어적으로 어떻게 제공하고 내부적으로 해결하는 방법 등을 연구해 보았다.

본 스케줄링 기법은 각 객체들의 사건들을 보드에 배정하는 작업으로, 슬롯(slot)은 사건 배정시 배정 목표의 최소 단위가 되며 보드는 연속된 슬롯들로 구성되며 이는 자원의 개념으로 이용된다.

본 논문에서는 OOP개념을 도입하여 객체의 특성을 정의한다. 문제 정의의 특성상 클래스간에는 계층을 가지고, 상위 클래스는 하위 클래스에 속성들과 메소드(method)들을 상속(inheritance), 다중 상속한다. 이러한 클래스들의 계층 구조를 이용하여 객체들의 구조를 모델링함으로서 실세계의 구조들을 표현하였다. 클래스 계층의 단말(leaf)에 객체가 등록되며 객체의 등록시 상위 클래스의 속성(property)을 상속하며 다중 상속도 가능하다. 스케줄링시 제약은 전역 제약과 지역 제약으로 구성된다. 보드에 대한 정의, 갯수등에 대한 정보와 객체의 다중 상속시 규칙, 모든 사건들에 대해 순서에 대한 정보(precedence)등을 전역 제약으로 사용하고, 각 객체가 가지는 보드의 스롯들에 대한 선호도(preference)를 지역 제약으로 사용한다.

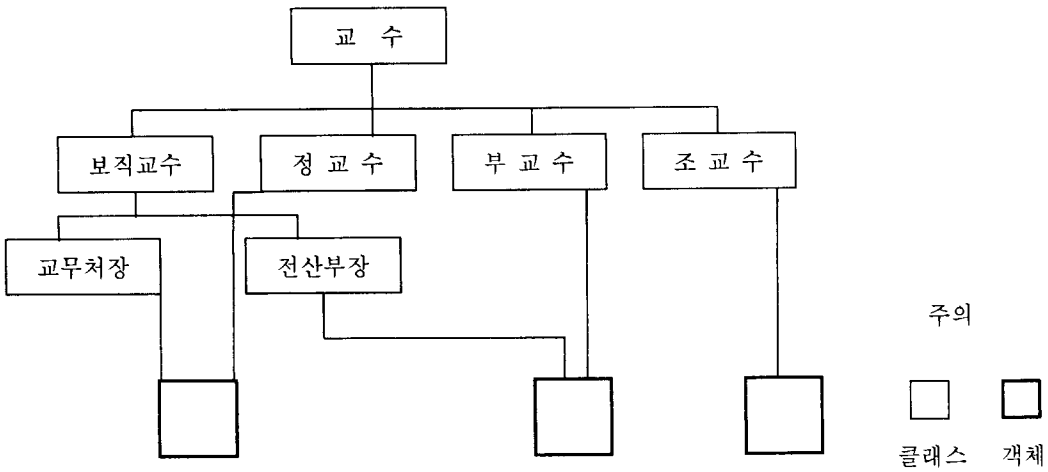
언급된바와 같이, 보드는 배정의 최소단위인 슬롯들로 구성된다. 슬롯 및 보드의 수와 보드구간의 수는 사용자가 정의할 수 있다. 보드 구간의



〈그림 1〉 보드의 예제(보드수=2, 보드구간수=2)

수는 시간상의 연속된 보드의 수이며, 보드수는 공간상의 수이다. 〈그림1〉에는 주 당 두 개의 보드가 시간적으로 2 차 연속으로 지속되며 각 보드의 4 일은 하루에 3 슬롯이 사용되고 마지막 날은 2 슬롯이 사용되는 예를 보여주고 있다. 실제 실제 보드가 그림같이 1 주와 반드시 매핑할 필요는 없으며, 임의의 형태와 행,열의 이름 및 각 행 및 각 열의 크기는 언어적으로 임의로 정의가능하다.

실세계의 문제에서는 각 객체가 속한 클래스를 우선 고려하여 어느 객체를 먼저 스케줄링할까를 결정하므로 이를 위해 객체지시형 개념이 사용되었다. 각 객체는 1 개 이상의 클래스에 속하며, 클래스의 속성과 메서드는 상속되며, 다중 상속도 가능하다. 〈그림2〉 에는 간단한 예가 도시되어 있다.



〈그림2〉 객체와 클래스의 예제

사건은 객체가 배정을 요청하는 단위인데, 사건명과 필요한 슬롯의 수로 구성된다. 사건당 연속 슬롯의 수는 광역적으로 제한되지만 사용자가 추가로 제한할 수도 있다. 객체가 갖는 사건의 최대 및 최소 수도 광역제한으로 제한될 수 있다.

광역제약은 스케줄링 중에 모든 독립된 객체에 영향을 주지만, 지역제약은 각 객체에만 영향을 주며 클래스의 속성의 형태로 존재하여 상속가능하다. 광역제약으로는 보드의 크기와 수, 보드의 구간 및 사건당 허용되는 연속된 슬롯의 수 등이 있다.

객체의 보드슬롯 선호도는 미리정의된 키워드의 형태로 표현가능하다. 키워드로는 'DEFINITELY' 에서 'DEFINITELY_NOT' 까지 9개로 구성되는데 내부적으로는 +1 에서 −1 까지의 선호도 강도로 처리된다. 선호도 강도가 +1 이거나 −1 이면 각각 반드시 만족되어야 하는 제약과 해당 슬롯이 배정되면 안되는 제약을 이르는데, 이 두 경우를 제한적 제약이라 부르며, 기타의 경우는 선호도 제약이라 부른다. 본 시스템에서는 선호도 제약을 중시하는데, 기본 동작방식은 4 장에서 언급되어 있다.

1.2 충돌 해결

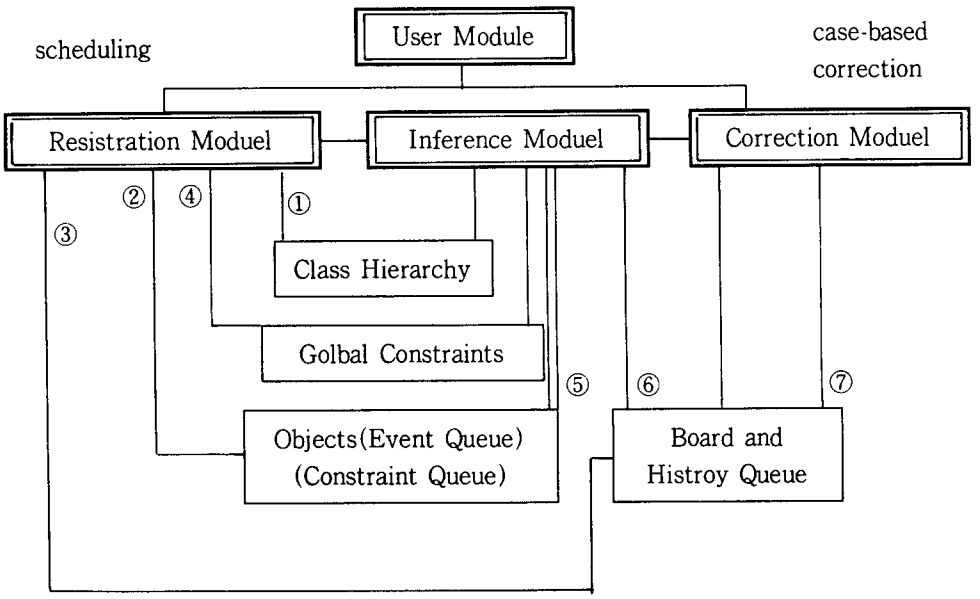
복잡한 문제 해결에서 백트래킹을 줄이고, 최적해를 빨리 구하려면 다음에 적용할 규칙의 선정이 매우 중요하다. 어느 규칙을 선정하는가하는 충돌 해결의 문제는 모든 최적해가 아닌 단 한 개의 준 최적해를 요하는 복잡한 스케줄링 문제에서는 매우 중요하다.

규칙에 근거한 우선순위 방식, 객체에 근거한 방식, 상태에 근거한 방식의 세 가지의 접근방식[Rich91]을 생각할 수 있다. 첫째 방식으로는 규칙의 위치에 따른 선정을 하는 프로로그 방식을 예로 들 수 있으며, 둘째 방식으로는 입력 문장이 가진 키워드(즉 객체) 중에서 가장 중요한 키워드를 선정하는 상담시스템 ELIZA[Weizenbaum66] 등이 있으며, 세째 방식은 최적우선 탐색등을 들 수 있다.

HI-SCHED 시스템에서는 기본적으로 객체에 근거한 방식을 사용한다. 공유되는 자원을 얻으려고 경쟁하는 객체 중에서, 객체의 우선순위에 의해 객체가 선정된다. 초기 우선순위는 소속 객체들에서 상속되는데, 객체의 사건이 배정됨에 따라 그 우선순위는 동적으로 변화한다.

2. 기본 시스템의 동작

〈그림3〉 에는 객체의 선호도로 구동되는 시스템의 기본적인 동작 모습을 보여 준다. 각 번호는 본 기법의 구동 시 운영되는 기본적인 작동 순서이며, 기본적인 동작은 다음절들에서 설명된다. 자세한 내용은 [Lee95]를 참조하라.



① 클래스 등록(class registration) ② 객체 등록(object registration)
③ 보드등록(board registration) ④ 전역제약등록(global constraint registration)
⑤ 제약큐의 조각화(constraint queue construction) ⑥ 주요 스케줄링(main scheduling)
⑦ 케이스에 근거한 교정(case-based correction)

〈그림 3〉 스케줄링 기법의 블럭 다이어그램

3. 객체의 등록

각 객체는 HI-SCHED 언어에서 〈그림4〉(a) 에서와 같이 표현되며, 등록된 직후의 객체는 그림 4〉(d)에서와 같이 사건 큐와 제약 큐를 갖는다. 사건 큐에 등록되는 사건은 사건의 이름과 필요한 슬롯의 수로 구성된다. 제약 큐는 2 개 이상의 상충된 슬롯선호도를 가질 경우 그 중 최소값을 슬롯의 슬롯 선호도로 하여 슬롯의 배정을 기다리는 선호도의 순으로 정렬되는 과정인 〈그림4〉(b)와 같은 조각화 과정을 거쳐 생성된다. 조각화 후에 객체가 보는 보드의 모습은 〈그림4〉(c)와 같다.

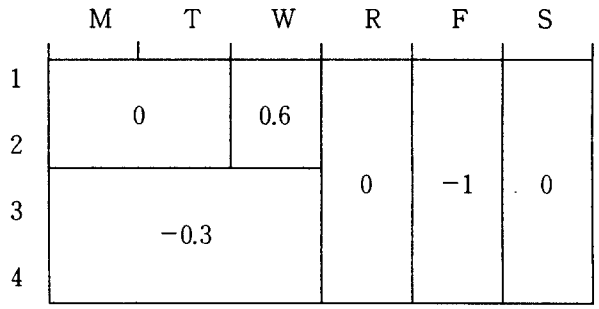
```
NAME STHONG : ASST_PROF;
  CONSTRAINT
    PROBABLY W 1-4, SLIGHT_NOT MTW 3-4,
    DEFINITELY_NOT F 1-4
  EVENT
    T_F 6, T_H 2;
```

(a) 스케줄링 언어로 작성한 객체의 예

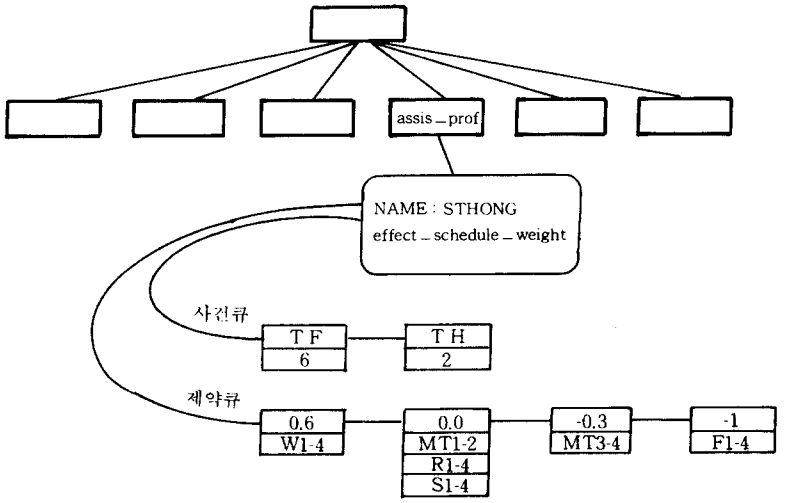
본래의 제약 큐 : (0.6, W1-4) → (-0.3, MTW3-4) → (-1, F1-4)

조각화된 제약 큐 : (0.6, W1-2) → (0, MT1-2 RS) → (-0.3, F1-4)
→ (-1, F1-4)

(b) 조각화 전 후의 객체의 제약큐



(c) 논리적으로 조각화된 보드



(d) 조각화 후의 객체의 표현

〈그림 4〉 객체의 표현

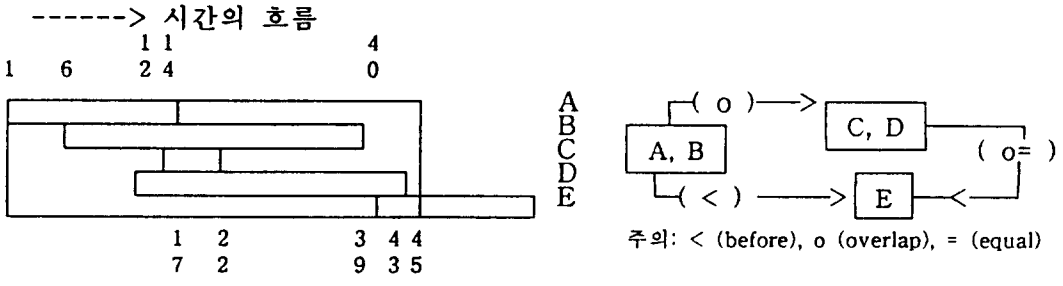
4. 전역 제약의 역할

각 객체의 스케줄 우선순위는 사건이 배정되면 감소하고, 백트래킹으로 사건이 취소되면 증가한다. 그 증감분은 전역제약으로 명시할 수 있다. 두개 이상의 클래스에 속하는 객체의 초기 스케줄 우선순위는 해당클래스의 우선순위의 합과 1 중 작은것으로하는 휴리스틱을 사용하는데, 그 방법의 구체적 명시를 위해서 전역제약을 사용할 수 있다. 보드 슬롯들을 위한 연속제약 등도 전역제약으로 선언된다.

5. HI-SCHED의 백트래킹 처리

백트래킹은 탐색공간을 탐색하는 체계적인 방법인데, 백트래킹을 개선하는 방법은 '뒤를 보는' 방법과 '앞을 보는' 방법으로 나누어 질 수있다. 뒤를 보는 방법은 지적 백트래킹이라 통칭될 수 있으며 프롤로그나 TMS 등에서 사용되는데, 과거의 정보를 이용하여 백트래킹의 지점을 선정한다. 앞을 보는 방법에서는 변수 또는 변수값 선택의 순서를 최적화하여 백트래킹 자체의 수를 감소시킨다.

HI-SCHED 에서는 기본적으로 '앞을 보는' 기법으로 분류할 수 있는 '동적 객체 스위칭'을 사용하여 백트래킹의 발생가능성을 원천적으로 줄이려고 노력한다. 그러나 사건 배정이 실패할 경우, 바로 전의 사건 대신에 실패에 가장 직접적인 영향을 끼친 사건을 선정하는 '뒤를 보는' 지적 백트래킹 기법도 사용된다. HI-SCHED 의 사용자는 사건 간의 시간 우선순위[Allen83]를 알렌이 정의한 13 개의 연산자를 이용하여 선언할 수 있다. 〈그림5〉(b)에는 사건간의 시간 우선순위의 예가 도시되어 있다. 〈그림5〉(a)와 같이 사건 A, B, C, D가 차례로 배정된 후 사건 E 의 배정이 실패하면 바로 전에 배정된 D 를 백트래킹 지점으로 고려하지 않고, 시간 우선순위 제약에 의해 E에 직접 영향을 미치는 B로의 백트래킹을 수행한다.



(a) 사건의 배정(E 의 배정 실패) (b) 시간 우선순위 제약

〈그림5〉 시간추론 방식에 따른 지적 백트랙킹용 사건의 선정

6. 케이스에 근거한 교정

인간의 스케줄링 방식의 또 다른 특성은 유사한 기 스케줄링을 기억 또는 기록하고 있다가 이를 근거하여 스케줄링한다는 점이다. 또한 이미 스케줄된 결과를 부분적으로 수정할 필요성도 있다. 따라서, 특정 사건의 취소와 삽입이 가능하여야 한다. 취소에는 해당 사건이후의 모든 사건을 연쇄 취소하거나 해당 사건만을 취소할 수도 있다. HI-SCHED 에서는 이러한 기능을 내장하고 있으나 자세한 내용은 본 논문의 흐름과 무관하므로 생략하기로 한다.

Ⅲ. 선호도 중심 스케줄링의 성능 향상

인간 방식의 복잡한 스케줄링 문제의 해결은 선호도 중심 해를 구하는 주처리와 주처리에서 나온 결과를 모든 요소를 고려하여 교정하는 후처리로 구성될 수 있다. 전 장에서 고려한 HI-SCHED 의 기본 기법을 주처리 방법으로 할 때, 주처리의 개선 방법과 후처리에 의한 개선방법을 고려할 수 있다.

1. Lookahead 를 이용한 동적 객체 스위칭 기법

객체의 동적 스위칭 시에는 다음에 고려할 객체는 객체 큐 상에 스케줄링 우선순위 순으로 분류되어 선택된다. 그러나 스케줄링 우선순위가 가장 높은 다음 객체만을 고려한 배정이 성능 저하를 가져올 수 있으므로 1 - 수 개의 Lookahead 크기(즉 미리 고려하는 객체의 수)를 사용하여 만족도를 개선할 수 있으면, 그 객체의 선택을 바꿀 수 있게하는 방법이다. 실험 결과 Lookahead 크기 1 만으로도 상당한 성능 저하를 가져올 수 있음을 알 수 있다. 다음의 예를 살펴 보자.

	월	화	수	목	금
1	0.5	0.5	0.5		
2	0.5	0.5	0.5		
3	−0.5	−0.5		0.2	0.2
4	−0.5	−0.5		0.2	0.2

(a) kim : 스케줄링 weight = 0.9, 필요 슬롯수 = 6

	월	화	수	목	금
1	1.0		0.3	0.3	
2	1.0		0.3	0.3	
3		−0.5	−0.5		
4		−0.5	−0.5		

(b) lee : 스케줄링 weight = 0.9, 필요 슬롯수 = 6

	월	화	수	목	금
1				0.5	0.5
2		−0.5	−0.5	0.5	0.5
3		−0.5	−0.5	1.0	
4				1.0	

(c) park : 스케줄weight = 0. 7, 필요슬롯수 = 6

〈그림 6〉 객체의 슬롯 선호도의 예제

	월	화	수	목	금
1	kim	kim	lee	lee	park
2	kim	kim	lee	park	park
3	lee			kim	lee
4	lee	park	park	kim	park

(a) Lookahead 크기 = 0 인 스케줄링으로 배정된 결과

	월	화	수	목	금
1	lee	kim	kim	lee	park
2	lee	kim	kim	lee	park
3	lee			park	kim
4	lee	park	park	park	kim

(b) Lookahead 크기 = 1 인 스케줄링으로 배정된 결과

〈그림 7〉 Lookahead 사용 및 비사용시의 스케줄링

각 객체의 만족도를 실재 배정된 슬롯들의 선호도의 합과, 경쟁이 없이 배정할 경우의 슬롯 선호도의 합의 비율이라고 정의하면, 〈그림 7〉(a)와 같이 배정된 경우의 각 객체의 만족도는 다음과 같다.

kim의 만족도 = (0.75*4 + 0.6*2) / (0.75*6) = 0.93

lee의 만족도 = (0.65*3 + 0.5*3) / (1.0*2 + 0.65*4) = 0.75

park의 만족도 = (0.75*3 + 0.5*3) / (1.0*2 + 0.75*4) = 0.75

즉, 3 객체 자체의 중요도를 동일하게 볼 경우의 평균 만족도는 (0.93 + 0.75 + 0.75) / 3 = 0.81이다.

〈그림 7〉(b) 와 같이 배정된 경우(즉 Look-ahead 크기 = 1)의 각 객체 kim, lee, park 의 만족도를 동일한 방식으로 계산하면 각각 0.93, 0.93, 및 0.9 가 되고 객체들의 평균 만족도는 0.92 이다.

본 예에서는 Lookahead 사용시의 성능 향상이 0.81 에서 0.92 로 11% 증가하였음을 알 수 있다. Lookahead를 고려한 시스템의 만족도 향상은 처리하려는 문제의 특성에 의존하긴 하지만, 선호도 중심 스케줄링에서 중요한 성능 향상 기법이 될 수 있다.

2. 주처리시 전역 제약의 추가 처리

전역제약으로 고려할 항목은 스케줄링의 성격 및 목적에 따라 다르며, 그 종류도 다양하다. 따라서 추가되는 제약은 모두 후처리에서 담당하도록 하는 것이 일반적인 해법으로 볼 수 있다. 그러나 다음과 같은 두가지 제약을 주처리 과정에서 추가하는 방법을 고려하여 간단한 실험을 시도하였다.

☞ Slot-in-Event Distribution Constraint
사건들을 원하는 요일(즉 column)들 기준으로 퍼지적으로 보드에 배정하기 위한 제약. 예를 들어 각 사건을 가능한한 MWF 에 또는 TR 에 분포하게 하는 제약

☞ Hours-per-Day Constraint
하루에(column을 뜻함) 배정되는 시간수(슬롯 수를 뜻함)를 퍼지적으로 처리하기 위한 제약 예를 들어 MTWRF 에는 각각 3 내지 5 시간을 S 에는 1 내지 2 슬롯을 원할 경우 사용.

가. Slot-in-Event Distribution Constraint column_list$^+$

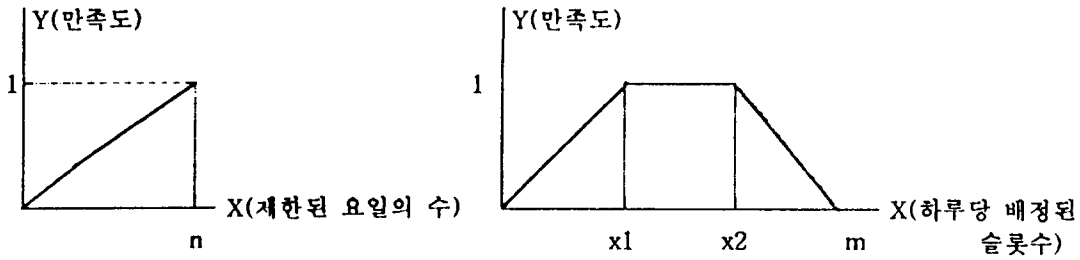
제한하려는 column 명들의 묶음마다 별도의 Con_set(i)를 구성하고, 나머지 column들로 Rem_set을 구성한다. (예를 들어, 보드내에서 가능한 column 명이 M, T, W, R, F, S인 경우 column_list 를 MW, TR 로 제한하면 Con_set(1)은 M, W로 Con_set(2)는 T, R로 구성되고, Rem_set은 F, S로 구성된다.)

각각의 Con_set[i]에 대응하는 만족도를 Y, 실제 제한된 row의 수를 X라 하면 〈그림 8〉(a) 에서와 같이 $Y = X/n$ 으로 표현할 수 있다. 단 n 은 제한된 요일들의 주어진 수이다.

Rem_set에 따른 만족도의 배정은 해당 사건이 모두 Rem_set에 속하는 경우에만 1을 배정하고, 그 이외의 경우에는 무시한다. 또한, Con_set이 두개 이상인 경우 서로 충돌이 생길 수 있는데, 그러한 경우에는 각각의 만족도중 최대값을 배정한다.

나. Hours-per-day Constraint [column_list hour_range]+

이 제약이 존재하면 이를 참조하여 각 column (c) 마다 Hours_set(c)를 구성한다. Hours_set(c)의 값은 c를 가진 column_list 에 해당되는 hour_range를 사용하는데, 그 꼴은 $x_1 \cdots x_2$ ($x_1 \le x_2$)를 갖는다. 선언되지 않은 경우는 물론 1…c의 슬롯수를 갖는다. 각각의 Hours_set[c] 에 대응하는 만족도 Y는 〈그림 8(b)〉와 같다. 구체적으로 해당 요일에 대해 제한슬롯의 수(가령 1시간이 단위일 수있음)가 x_1에서 x_2사이이면, 만족도는 1이며, x_1보다 작은 X값에 대해서는 $Y = X/x_1$를 적용하며, x_2보다 큰 X값에 대해서는 $Y = (m-X)/(m-x_2)$를 적용한다. 이 때 m은 하루에 배정가능한 총 슬롯의 수를 나타낸다.



(a) Con−set(i)에 대한 만족도 (b) Hours_set(c) = $x_1 \cdots x_2$에 대한 만족도

참고 : 위의 만족도는 각각의 i 및 c에 대해 별도로 존재함

〈그림 8〉 추가 제약에 따른 생성 집합의 만족도

3. 간단한 후처리의 추가

후처리의 경우는 시뮬레이티드 어닐링과 같은 임의화 방법에 근거한 전역적 방법과 주처리의 결과를 토대로 지역 최적화를 추구하는 방법으로 나눌 수 있는데, 주처리의 충실도를 고려해 볼 때 후자가 적합하다. 전자의 경우는 적용시 전처리의 결과가 무시되며, 시간적인 문제 등을 고려해 볼 때 바람직하지 못하다. 임의화 방법의 결과가 일반적으로 좋은 결과를 기대할 수 없는 또다른 이유로는, 모든 요소의 중요도를 배정하는 어려운 문제에 봉착한다는 점과 정확한 묘사가 사실상 불가능한 목적 함수값을 전역 최적화의 대상으로 고려하게된다는 점을 들 수 있다.

Ⅳ. 구현 결과 및 비교 분석

1. 제약 추가에 따른 스케줄링 알고리즘

각 객체들이 동적으로 스위칭 하면서 사건들을 배정하며, 추가된 전역 제약도 고려하는 알고리즘은 다음과 같다. 여기서 각 개체는 부모 클래스에서 상속받은 스케줄링 우선 순위에 따라 정렬되어 있다고 하자. history 큐는 스케줄링의 결과에 대해 정보를 가지는 큐라 하고, 백트랙킹 큐는 사건 배정 실패시 임시로 저장하는 큐를 의미하며, 각각의 set들은 앞에서 언급한 것과 같은 기능을 가진다.

```
/*필요시 각 객체의 슬롯헌호도를 평균 0 근처가 되도록 상대값으로 변환한 후 다음을 수행함*/
REPEAT
  IF 백트랙킹 큐 <> nil THEN
     백트랙킹 큐에서 사건 하나를 꺼내 e라 하자;
     사건 e에 해당하는 객체를 o라 하자;
  ELSE
    REPEAT
     객체 agenda상의 다음 객체를 o라 하자;
     IF o = nil STOP;
     WHILE o 의 사건 큐 = nil BEGIN
       객체 agenda상의 다음 객체를 o라 하자;
     END;
     o의 사건 큐의 다음 사건을 e라 하자;
     UNTIL e <> nil
  ENDIF
  /* 이제 사건 e와 해당 객체 o가 선정되었다 */
  o의 제약큐를 가지고 사건 e를 전역 제약과 지역 제약에 어긋나지 않도록 현 보드에 등록된 내용과
  겹치지 않게 배정을 시도한다;
  IF 배정에 실패 THEN
       현 객체의 사건 e를 백트랙킹 큐에 넣고 적절히 history큐에서 하나의 사건을 사건 간의 시간
       제약을 고려 선택하고 이를 취소시킨다;
       /* 바로 이전의 사건을 해제시키지 않고, 실패의 원인을 조사한다 */
       history 큐에서 꺼낸 사건은 취소된 사건 수만큼 '유효스케줄링weight'를 증가시킨다
  ELSE          /* 배정에 성공 */
       보드 history 큐에 객체 o의 사건과 배정된 슬롯에 대한 정보를 등록한다; Hours-set에 배정
       되는 사건의 정보를 등록한다;
       IF 하나의 사건 배정이 모두 끝남 THEN Slot-in-Event- Distribution 프로시져를 부른다;
       객체 o의 '유효스케줄링weight'를 전역제약에 맞게 감소시킨다;
       새로운 '유효스케줄링weight'에 맞게 o의 위치를 객체 agenda상에서 이동한다;
  ENDIF
UNTIL 모든 사건들의 배정이 끝남;
각 보드에 대해 Hours-per-Day_Handler 프로시져를 부른다;
```

```
/* 사건의 슬롯들이 분포하는 요일을 제약하는 프로시져 */
PROCEDURE Slot-in-Event_Distribution
BEGIN
/* 광역적으로 정의되어 있는 임의개의 Con-set 와 1개의 Rem-set을 편의상 set부르자 */
 REPEAT
   IF 해당 사건 e의 모든 슬롯이 어떤 set에 들어 있음 THEN      RETURN;
   s := e의 이동안된 슬롯 중 최저선호도 갖는 슬롯;
   s가 속한 set의 row에 속하지 않는 빈슬롯(empty slot) 중 o의 최대선호도 갖는 슬롯의 선호도가
   s의 선호도 이상이면 슬롯 이동;
/* 교정슬롯선호도 ≡ 슬롯을 차지한(또는 차지할) 객체의 '유효스케줄링weight' * 슬롯선호도
          * 그림 8(a)에서 정의된 만족도 */
FOR s := e의 이동 또는 교환안된 슬롯 중 최저선호도 갖는 슬롯
 REPEAT
    s' := s가 속한 set의 row에 속하지 않는 검사안된 찬슬롯(full slot) 중 o의 최대선호도 갖는 슬롯;
    IF s'이 존재안함 THEN    RETURN; /* 교환 포기하고 끝냄 */
    s'의 교정슬롯선호도가 s 의 교정슬롯선호도 이상이면 두 슬롯을 교환함;
     /* 슬롯교환은 동일 객체간 또는 타 객체간에도 허용할 수 있음 */
    IF 해당 사건 e의 모든 슬롯이 어떤 set에 들어 있다 THEN      RETURN;
    UNTIL 교환이 됨;
  ENDFOR
END;
/* column(즉 day)당 슬롯(즉 hour)수 제약 프로시져 */
PROCEDURE Hours-per-Day_Handler
BEGIN
 FOR c := 보드상의 각 column(요일에 해당됨)
   X := c 에 배정된 찬슬롯(full slot)의 수;
   (X1 , X2) := Hours_set(c);
   IF X ∈ (X1, X2) RETURN;
/* 교정슬롯선호도2 ≡ 교정슬롯선호도 * 그림 8(b)에서 정의된 만족도 */
   IF X<X1 THEN /* 다음은 Slot-in-Event_ Distribution과 유사하며 간단히 기술한다 */
     c 이외의 임의column의 검사안된 빈슬롯 중 s 소유 객체기준의 교정슬롯선호도 2를 사용하여 그
          값이 감소하지않는 경우 column c로의 이동을 시도한다;
   ELSE /* X>X2인 경우 */
     s := column c의 검사안된 슬롯중 교정슬롯선호도2가 가장 낮은 슬롯;
     s를 c이외의 임의column의 검사안된 빈슬롯으로 이동시, s소유 객체기준의 교정 슬롯선호도 2의
     값이 감소하지 않는 경우 이동을 시도한다;
     실패시에는 교정슬롯선호도 2가 가장 높은 슬롯(s')과 교환시 교정슬롯선호도 2가 감소하지 않는
     경우 교환을 한다.
     ENDIF
   ENDFOR;
END;
```

2. HI-SCHED 언어를 이용한 코딩의 예제

HI-SCHED 언어를 이용한 코딩의 예제 본 절에서는 HI-SCHED 를 이용한 프로그램의 예를 다룬다. 추가적인 전역제약은 고딕체로 표시되어 있다. 자세한 내용은 여기서 생략한다.

```
CLASS DECLARATION
    CLASS PROTO_PROF;
    CLASS FULL_PROF : PROTO_PROF;
    CLASS ASSOC_PROF : PROTO_PROF;
    CLASS ASSIS_PROF : PROTO_PROF;
    CLASS INSTRUCTOR : PROTO_PROF;
    CLASS LECTURER : PROTO_PROF;

GLOBAL CONSTRAINT DECLARATION
    BOARD CONSTRAINT
      1 INTERVAL;
      M 1-8, T 1-8, W 1-8, H 1-8, F 1-8, S 1-8;
    INHIBIT CONSTRAINT 6;
    CONTIGUOUS CONSTRAINT 2;
    SLOT_NO CONSTRAINT 0-15;
    SLOT_IN_EVENT_DISTRIBUTION CONSTRAINT MWF
    HOURS_PER_DAY CONSTRAINT M-S 3.5

  OBJECT DECLARATION
    NAME KDHONG : FULL_PROF;
      CONSTRAINT
        PROBABLY M 1-4, SLIGHTLY TWHF 5-8, DEFINITELY_NOT S 1-4;
      EVENT
        T_A 6, T_E 4;
    NAME BJBANG : ASSOC_PROF;
      CONSTRAINT
        CERTAINLY T 1-4, SLIGHT MWF 1-4, PROBABLY_NOT W 5-8;
      EVENT
        T_B 6;
    NAME KDGO : ASSIS_PROF;
      CONSTRAINT
        PROBABLY WS 5-6, SLIGHTLY_NOT MTW 3-4, DEFINITELY_NOT H 1-4;
    EVENT
      T_C 6;
    NAME SJHONG : INSTRUCTOR;
      CONSTRAINT
        CERTAINLY H 1-4, SLIGHTLY MTW 1-8, DEFINITELY_NOT F 1-4;
    EVENT
      T_D 6;
END.
```

3. 스케줄링 결과

SCHED 언어를 이용한 간단한 예제에 대한 스케줄링의 결과는 새로운 전역제약을 사용하지 않은 경우와 사용한 경우에 대해 〈그림 9〉의 (a)와 (b)에 표시되어 있다.

	M	T	W	R	F	S
1	T_A	T_B	T_B	T_D	T_B	
2	T_B	T_B	T_B	T_D		
3	T_E	T_D				
4	T_E	T_B		T_D		
5	T_C	T_E	T_C			T_C
6	T_C	T_E	T_C			T_C
7	T_D	T_A	T_A			
8	T_D	T_A	T_A			

(a) 추가된 전역제약을 사용하지 않은 경우

	M	T	W	R	F	S
1	T_A	T_B	T_B	T_D	T_B	
2	T_B	T_B		T_D	T_B	
3	T_E	T_D			T_C	
4	T_E	T_B				
5	T_C	T_D	T_C	T_D	T_E	T_C
6		T_D	T_C	T_A	T_C	
7			T_E	T_A		
8			T_A			

(b) 추가된 전역제약을 사용한 경우

	M	T	W	R	F	S
1	T_A	T_B	T_B	T_D	T_B	
2	T_B	T_B		T_D	T_B	
3	T_E	T_D			T_C	
4	T_E	T_B				T_D
5	T_C	T_D	T_C	T_D	T_E	T_C
6			T_C	T_A	T_A	T_C
7			T_E	T_A		
8			T_A			

(c) 단순처리를 적용한 후

〈그림 9〉 처리 완료된 보드의 모습

4. 후처리

새로운 두 개의 전역제약이 추가된 경우의 스케줄 결과를 이용하여 후처리를 시도하였다. 후처리 과정에서는 SCHED 언어를 실제로 적용하려는 영역에서 필요한 모든 추가적인 제약들을 고려할 수 있어야 한다. 주처리에서 얻은 만족도는 결국 다음절에서의 OSD 를 개선하기위한 방법인데, 후처리에서는 기타의 제약들을 BESD 의 형태로 표현하여, 이들 모두를 고려하는 TSSD 를 최적화하는 시도가 바로 후처리라고 할 수 있다.

여기서는 단순한 단순한 후처리 기법으로 빈 슬롯으로의 이동 만을 고려하였다. 이 방법은 슬롯을 채우는 율이 비교적 높지 않은 경우에 적합하다고 사료된다. 이와 달리 무작정 임의의 빈 슬롯으로의 이동을 시도하는 모의 어닐링(simulated annealing) 등의 방법을 시도할 경우 주처리의 의미가 없어지므로, 최적화 기법은 기존의 주처리를 인정하고 지역 최적화를 통해 추가적인 기타 제약이 있을 경우 이용하는 것이 바람직하다. 여기서는 슬롯의 선호도가 음수인 곳을 피하고 임의의 빈 슬롯으로의 이동이 가능하게 하는 기법을 사용하였다. 〈그림 9〉(c)에 나타난 스케줄링 실험 결과는 〈그림 9〉(b) 에서 T의 6째 슬롯의 T-D만이 S의 4째 슬롯으로 이동했다(즉, 단 하나의 슬롯 이동만이 있었다.)

5. 결과의 평가 기법

선호도에 의한 배정기법을 사용해도 결과를 평가할 필요가 있으며, 이를 이용하여 후처리의 최적화 방식으로 사용하거나 주처리의 개선에 이용할 수 있다. 다음과 같은 메저(measure) 들이 정의되어 이용되었다.

전체 시스템 만족도 TSSD(Total System Satisfaction Degree) 는 객체 만족도 OSD(Object Satisfaction Degree) 와 보드및 사건 만족도 BESD(Board and Event Satisfaction Degree) 의 무게갖는 평균(weighted average)으로 정의된다. 단 값은 0.5보다 큰데 이는 OSD 를 더 중요시하기 위한 의도이다. 보드 및 사건 만족도는 객체만족도 이의의 요소를 최종 스케줄링 결과 분석 및 개선을 위해 사용하는데, 특정스케줄링과 관련된 모든 요소를 필요시 추가로 고려할 수 있다.

$$\mathbf{TSSD} = \alpha * \mathrm{OSD} + (1-\alpha) * \mathrm{BESD} \quad (\text{단, } \alpha > 0.5)$$

OSD와 BESD 각각의 정의는 다음과 같다.

$$\mathbf{OSD} = (1/n) \sum_{i=1 \cdots n} S_o(i) / S_o'(i)$$

n : 총 객체의 수

$S_o'(i)$: 각 객체가 경쟁이 없을 경우 보드에 대해 가질 수 있는 최대 만족도

$S_o(i)$: 객체들이 경쟁하고 나서 배정받았을때의 만족도

$$\mathbf{BESD} = \beta * [(1/m) \sum_{j=1 \cdots m} S_e(j)] + (1-\beta) * [(1/l) \sum_{k=1 \cdots l} S_b(k)] \quad (\text{단, } \beta > 0.5)$$

m : 총 사건의 수

$S_e(j)$: 각 사건(event)에 대한 슬롯 분포만족도

l : 총 보드의 수

$S_b(k)$: 각 보드 별 분포 만족도(추가사항을 고려한 정의도 가능)

예제 에서 사용된 슬롯 비율은 58%이다. (가능한 슬롯수는 48개, 사용된 슬롯은 28개)사용한 α와 β는 모두 0.6으로 하였다.

후처리로 빈 슬롯으로의 이동기법을 사용한 결과 하나의 슬롯만 이동하였을때 가장 최적의 결과치를 얻을수 있었고, OSD값은 도표에서 보듯이 약 0.03정도 떨어졌고, TSSD값은 약 0.03정도 상승되었다. 새로운 제약들의 추가로 만족도 TSSD의 급격한 상승을 가져왔으며, 단순 후처리

	새 제약이 추가되지 않을 때	새 제약이 추가될 때	후처리
OSD	0.94	0.94	0.91
TSSD	0.77	0.90	0.93

만으로도 TSSD의 증가(0.03)를 얻을 수 있었다. 여기서 후처리는 TSSD 의 개선을 위해 시도되므로, 객체만족도인 OSD의 일부 감소는 예측된 결과이다.

V. 결론 및 연구 방향

본 논문에서는 인지과학적 접근에 근거한 선호도중심 스케줄링 언어로 개발한 HI-SCHED 의 처리 기법들을 중심으로 성능 분석 및 성능 향상 기법을 다루고 있다.

컴퓨터로 사실상 최적해를 구할 수없는 복잡한 스케줄링 등의 문제에서는 주처리와 후처리로 이루어진 두 단계의 문제 해결 방법론이 적합한 문제 해결의 방식으로 적합하다. 주처리는 스케줄링에서 가장 중요한 객체의 보드 슬롯 선호도를 객체지시형 환경을 고려하는 동적 객체 스위칭 방식으로 해결하여 사용가능한 해를 제시한다. 이런 주처리의 결과를 판단하는 메저가 제시되었으며, 문제의 성격에 부합하는 많은 수의 요소를 복합처리해줄 후처리로는 전역적인 방법보다는 지역적인 방법이 적합하다고 판단된다. 추가적인 전역제약을 고려하는 메저도 정의되었으며, 특정 예에 대해 그 값이 제시되었다.

이러한 방법론은 복합적인 스케줄링의 일반적인 해법으로 사용가능하며, 그 성능도 점검하여 그 효능을 확인하였다. 미리 1-수개의 객체를 미리 보고 동적 객체 스위칭을 수행하는 방법도 연구되었다. 실험 결과 Lookahead 크기가 1 인 경우로도 큰 성능향상을 기대할 수 있음을 보여주고 있다. 동적 객체 스위칭시 사건이 배정된 객체의 스케줄 weight 의 감소방법과 후처리의 수준 등을 더욱 효율적으로 고려하는 기법의 연구가 필요하다. 또한 표준화된 복합 스케줄링 언어를 통한 접근방식에 대한 일부 논란이 있을 수 있으나, 이를 통해 유사한 영역의 문제에 대한 표준화된 접근은 스케줄링 유형 및 분류에도 도움을 줄 수 있을 것으로 판단된다.

참 고 문 헌

[Aiba88] Aiba, A. and et al., "Constraint logic programming language CAL," Proceeding of the International Conference on Fifth Generation Computer Systems, Ohmsha Publishers, Tokyo, pp. 263-276, 1988.

[Allen83] Allen, J. F., "Maintaining Knowledge about Temporal Intervals," Communications of ACM 26(11), 1983.

[Allen&Koomen83] Allen, J. F. and Koomen, J. A., "Planning using a temporal world model," 8th Int. Joint Conf. Artificial Intelligence, Aug. 1983.

[Chapman87] Chapman, D., "Planning for conjunctive goals," Artificial Intelligence 32(3), 1987.

[Cohen90] Cohen, J., "Constraint Logic Programming Language," Communication of the ACM, July 1990.

[Colmerauer90] Colmerauer, A., "An Introduction to Prolog III," Communications of the ACM 33(7), pp. 69-90, 1990.

[Dechter88] Dechter, R. and Pearl, J., "Network-based heuristics for constraint satisfaction problems," Aritificial Intelligence 34, 1988.

[Dincbas88] Dincbas, M. and et al., "The constraint logic programming language CHIP," Proceeding of the International Conference on Fifth Generation Computer Systems, Ohmsha Publishers, Tokyo, pp. 693-792, 1988.

[Ernesto93] Ernesto, M. M., and Joao, P. M., "An AI-based approach to crew scheduling," 9th Conference on AI for Applications, March1-5, 1993.

[Fikes&Nilsson71] Fikes, R. E. and Nilsson, N. J., "STRIPS: A new approach to the application of theorem proving to problem solving," Artificial Intelligence 2, pp. 189-208, 1971.

[Fox91] Fox, M. S., and Zweben, M., "Knowledge based Scheduling," AAAI-91, 1991.

[Fukumori80] Fukumori, K., "Fundamental Scheme for Train Scheduling," A. I. Me3mo 596, AI Lab, MIT 1980.

[Hammond86] Hammond, K., "Chief: a model based planning," Proc. AAA-86, 1986.

[Lee94] Lee, K. C., and et al., "A Human Preference-Driven Object-Based Scheduling Technique," pp. 91-96, IEEE TENCON, 1994.

[Lee95] Lee, K. C., "Object-Based time Scheduling, Focusing on Object Preference Constraints," KOSEF 931-0900-017-2 Final Report, April 1995.

[McDermott82] McDermott, D., "A Temporal Logic for Reasoning about Processes and Plans," Cognitive Science 6, pp101-155. 1982.

[Miller83] Miller, D., "Scheduling Heuristics for Problem Solvers," Research report 264, Comp. Sci. Dept., Yale Univ., 1983.

[Nilsson80] Nilsson, N. J., Principles of Artificial Intelligence, Palo Alto, CA: Morgan Kaufman, 1980.

[Rich91] Rich, E., and Knight, K., Artificial Intelligence (2 ed.), McGraw-Hill Inc., 1991.

[Sacerdoti74] Sacerdoti, E. D., "Planning in a hierarchy of abstraction spaces," Artificial Intelligence 5(2), pp. 115-135, 1974.

[Sacerdoti75] Sacedori, E. D., "Planning in a hierarchy of abstraction spaces," Artificial Intelligence 5(2), 1974.

[Stefik81a] Stefik, M., "Planning with constraints (MOLGEN: Part 1)," Artificial Intelligence 16(2), 1981.

[Sussman75] Sussman, G. J., A Computer Model of Skill Aquisition, Cambridge, MA: MIT Press, 1975.

[Vere81] Vere, S., "Planning in Time: Windows and Durations for Activities and Goals," Technical Report. Jet Propulsion Lab., 1981.

[Weizenbaum66] ELIZA-a computer program for the study of natural language communication between man and machine. Communications of the ACM 9(1):36-44, 1966.