IE Interfaces
Vol. 12, No. 3, pp. 362-373, September 1999.

Constraint Programming을 이용한 자원제약 동적 다중프로젝트 일정계획†

이화기 · 정제원

인하대학교 산업공학과

Resource Constrained Dynamic Multi-Projects Scheduling Based by Constraint Programming

Hwa-Ki Lee · Je-Won Chung

Resource Constrained Dynamic Multi-Projects Scheduling (RCDMPS) is intended to schedule activities of two or more projects sequentially arriving at the shop under restricted resources. The aim of this paper is to develop a new problem solving method for RCDMPS to make an effective schedule based by constraint programming.

The constraint-based scheduling method employs ILOG Solver which is C++ constraint reasoning library for solving complex resource management problems and ILOG Schedule which is a add-on library to ILOG Solver dedicated to solving scheduling problems. And this method interfaces with ILOG Views so that the result of scheduling displays with Gantt chart.

The scheduling method suggested in this paper was applied to a company scheduling problem and compared with the other heuristic methods, and then it shows that the new scheduling system has more preference.

1. 서 론

본 연구는 금형 주문생산업체인 S사의 실제 일정계획 문제를 예로 하였으며, 이 회사의 일정계획의 특징으로는 하나의 주문에 대한 생산일정을 프로젝트 네트워크로 분해하여 각 공정의 선행관계와 설비제한을 만족시키면서 전체 공정의 일정을 수립하고 이를 토대로 공정을 진행시키고 있다. 또한 이러한 주문들은 정해진 시간이 없이 비정기적으로 수주됨으로써 새로운 주문이 이루어질 경우 제한된 설비에 맞추어 이후 모든 프로젝트들의 일정을 다시 세워야 하는 문제들이 발생한다. 따라서 본 연구에서는 이러한 문제를 해결하기 위한 효율적인 해법을 제시하려 한다.

자원제약 프로젝트 문제는 일반적으로 고정되어 있는 자원을 제약으로 하여 해당 프로젝트의 완료기간을 최대한으로 줄이는 데 그 목적을 두고 수행된다. 이러한 문제들은 주공정 분석을 이용하여 프로젝트의 완료기간을 산정하는 일반적인 프로젝트 문제와는 달리 프로젝트의 각각의 작업들의 수행시에 이용할 수 있는 자원의 양이 제한되어 있어 동시에 각각의 작업들이 같은 자원을 필요로 할 때, 각각의 작업들에 대한 우선순위를 결정해 주어야 하는 일정계획 문제가 된다.

자원제약 다중 프로젝트 문제는 프로젝트가 두 개 또는 그 이상의 숫자로 발생하여 각 프로젝트의 네트워크에서 작업의 선행관계와 자원 이용량을 만족시키면서 각 작업의 완료시간을 결정하는 것이다. 여기서 각각의 프로젝트는 여러 개의 활동(activity)들로 이루어져 있고 각각의 활동은 여러 종류의 자원을 이용해서 수행된다. 즉, 자원제약 다중 프로젝트 스케줄링 문제는 크게 주어진 자원의 효율적인 사용을 고려한 자원평준화(resource-leveling) 문제와 다중 프로젝트 수행기간(make-span)의 최소화 문제로 나누어질 수 있다.

† 이 연구는 1998년도 인하대학교 연구비 지원에 의해 수행되었음.

다중 프로젝트 스케줄링 문제는 프로젝트의 수가 커지고 이들을 구성하고 있는 활동의 수가 매우 많아지면 해석적 방법에 의해서 해를 구하기가 어렵다. 문제의 최적 해를 얻어내기 어려운 이유로 인해 문제의 해결을 위해 제시되고 있는 방법들이 heuristic 기법을 도입한 작업배분규칙이다.

작업배분 우선순위 규칙들은 지금까지의 연구에서 많은 것들이 발표되었으며, 자원제약에 대한 다중 프로젝트에 대한 연구에서 모든 프로젝트의 완료 시간을 최소로 하기 위한 규칙으로 SASP(Shortest Activity from Shortest Project, 가장 짧은 프로젝트에서 가장 짧은 공정시간을 갖는 작업을 우선), MINLFT(Minimum Latest Finish Time, 주공정 분석에 의해 얻어진 가장 늦은 완료시간을 갖는 프로젝트에서 가장 짧은 공정시간을 갖는 작업을 우선), MINSLK(Minimum Job Slack, 주공정 분석에서 얻어진 가장 늦은 시작가능시간과 가장 이른 시작가능시간의 차를 슬랙으로 하여 이것이 가장 작은 작업을 우선) 등이 좋은 결과를 가져온다고 보고되고 있다(Dumond and Marbent, 1998).

위의 연구결과들은 모든 프로젝트가 동시에 발생한다는 가정하에서의 배분규칙들로 본 연구에서 다루려는 새로운 프로젝트들이 비정기적인 시간간격을 두고 수주되는 동적인 상황에서는 바로 적용할 수가 없다. 이러한 경우의 문제는 자원제약 동적 다중 프로젝트 일정계획문제로 불리며, 이에 대한 연구결과는 미미하다.

Kurtulus and Davis(1982)는 새로운 프로젝트가 발생하는 시점 이후로 기존 프로젝트의 나머지 공정과 새로운 프로젝트에 SASP규칙과 FIFO(시간적으로 가장 먼저 발생되는 작업을 우선)를 적용한 결과 총 프로젝트들의 완료시간을 최소화하는데 FIFO규칙이 우위에 있다고 보고하고 있다. 이화기와 하승진(1996)은 새로운 프로젝트 시점 이후마다 MINLFT규칙을 적용한 결과 최소 완료시간면에서 FIFO나 SASP보다 우수하다고 하였다. 한편 이화기와 윤종준(1997)은 FIFO에 의해 구해진 초기해에 Tabu Search에 의한 삽입이동방법을 이용하여 최선 이웃해를 생성하는 방법을 제시하였다.

본 연구에서는 자원제약하의 동적인 상황의 다중 프로젝트 일정계획 문제에 대해서 인공지능방법의 일종인 Constraint Programming에 기반을 둔 C++ 라이브러리 ILOG Schedule과 Solver를 이용하여 현재까지의 모든 프로젝트의 총처리시간(makespan)을 최소로 하는 일정계획을 생성하고 아울러 새로운 재일정이 요구되는 경우 방법을 반복하는 새로운 일정계획기법을 제시하고자 한다. 또한 본 연구에서 제시되는 일정계획기법과 기존에 제안된 휴리스틱 규칙을 비교하여 새로운 기법의 우수성을 제시하려 한다.

2. Constraint Programming

Constraint Programming기법은 인공지능기법의 일종으로 제약조건을 만족하는 해를 찾아내는 방법을 가리킨다. 즉, 제약조건을 가지고 문제를 모델링한 후 여러 가지 인공지능(AI)기법이나 OR기법들을 동원하여 해를 구하고자 하는 방법이다.

기본적으로 이 기법은 모든 가능해 집합을 나열하여 가능해나 최적해를 찾아내는 것이다. 이 기법의 단점은 해를 찾아내는 데 걸리는 검색시간이 오래 걸린다는 점이다. 그래서 이 기법의 핵심은 어떻게 검색시간을 줄일 수 있느냐는 점에 있으며, 주로 적용되는 검색기법으로는 Forward-checking, Back-checking, Back-tracking들을 들 수 있다.

Constraint Programming에 대한 연구는 1980년대 초 Mark Fox에 의해 시작되었으며, 이를 적용한 최초의 시스템은 ISIS로 스케줄링문제에 이용되었다(Claude, 1994). 그 후 개발된 효과적인 Constraint Programming 언어로는 PROLOG III, CHIP, ILOG Solver 등을 들 수 있다. 이러한 언어는 제약식 언어의 독특한 이론에 의하여 효과적으로 표현되고 수행된다. 하지만, 이를 효과적으로 이용하기 위해서는 이의 배경이론뿐만 아니라 해결하고자 하는 문제의 특수한 구조에 대해서도 고려하여야 한다(Douglas and Stephen, 1995).

이러한 Constraint Programming을 이용한 시스템의 기본적인 특징은 첫째, 제약식의 정의부분과 Constriant Propagation법, 문제해결을 위한 검색알고리즘 부분을 서로 분리하고 있다. 둘째, 각 제약식은 가능한 한 국부적으로 Propagation되도록 한다. 즉, 선언부에서 선언된 제약식만을 가지고 불필요한 도메인을 Propagation을 통해 줄인 다음, 경영과학이나 AI기법들을 이용한 적절한 검색알고리즘을 이용하여 해를 구하게 된다.

Constraint Propagation은 가능해를 찾기 위해 Cutting Constraint방법을 통해 해를 분할하고 분할된 부분이 가능하지 않은 경우에는 더 이상 고려하지 않는다. 이러한 과정을 반복적으로 거쳐 가능해를 구하게 된다. Cutting Constraint의 방법은 가능해를 만족하는 범위 내에서 고려하는 범위의 집합을 계속 부분집합으로 나누어 가능하지 않은 부분집합은 제거하여 고려하는 문제의 범위를 줄여나가는 방법이다.

2.1 ILOG Solver

ILOG Solver는 최적화 문제에 적합한 C++ 프로그래밍 라이브러리이다. Solver의 주요한 특징은 Constraint Programming을 OOP(객체 지향 처리) 환경으로 통합한 것이다. 결과적으로 ILOG Solver의 중요한 관점은 단지 Scalar 변수가 아닌 Object에 대해 Constraint Programming을 직접 표현 가능하게 한 것이다.

2.1.1 ILOG Solver의 특징

- 정수 도메인, 실수 도메인, Symbol 값 및 Object 도메인, 정수집합 혹은 Symbol값 도메인에 대해서 최적화되도록 구현되어 있다.
- 제약조건을 Propagation하는 동안에 도메인을 줄이기 위해서 Arc Consistency 알고리즘을 사용한다.

• Object-Oriented Programming을 지원한다

• 제약조건 Propagation은 제약조건을 위반하는 의사결정을 방지한다. ILOG Solver는 스케줄링에 대한 의사결정이 제약조건을 위반하면 이 의사결정을 이전의 상태로 복구하는데, 이렇게 복구하는 것을 Backtracking이라고 한다.

• Prolog를 기반으로 하는 다른 프로그래밍 언어와는 다르게 Backtracking을 선택적으로 사용할 수 있기 때문에 nondeterministic algorithm과 deterministic algorithm을 쉽게 통합할 수 있다.

• 빠른 속도로 다양한 형태의 스케줄링 알고리즘을 생성하고, 여러 가지로 변경하여 실험하는 것을 가능하게 한다.

• 스케줄링을 생성하기 위해 필요로 하는 탐색의 횟수를 줄인다. 스케줄링의 의사결정이 만들어질 때마다 추론하기 위해서 이용되지 않은 제약조건을 이용하고, 의사결정과 불일치하는 대안을 찾아 탐색영역을 줄인다.

• 문제의 풀이와 문제의 표현이 분리되어 있다. ILOG Solver를 통한 알고리즘의 구현은 스케줄링 제약조건과는 독립적으로 탐색 Logic을 다룬다.

이러한 관점에서 ILOG Solver는 새로운 언어라기보다는 C++에서 유용한 Object와 라이브러리이다. Solver는 자체의 독특한 문법, 자료구조, 컨트롤구조, 입출력 함수들을 가지는 새로운 언어는 아니다. Solver는 제약조건을 관리하기 위해 필수적인 라이브러리를 조합함으로써 OOP를 확장한 것이다.

2.1.2 ILOG Solver의 기본적인 알고리즘

ILOG Solver안에서는 모델의 의사결정변수는 제약조건변수로 표현된다. 또한 각각의 제약조건변수는 가능한 값의 집합으로 고려중인 변수의 범위인 도메인(domain)을 갖는다. 제약조건을 만족하면서 제약조건변수의 "적합한" 값을 얻기 위해 탐색공간(search space)이 필요한데 그 탐색공간은 제약조건변수들의 모든 할당 가능한 값으로 이루어진다.

탐색과정은 트리구조로 표현될 수 있는데 트리의 노드(node)는 의사결정변수를 나타내고 트리의 가지(branch)는 이러한 변수들의 가능한 값을 나타내며, ILOG Solver가 노드에서 가지를 쳐 나갈 때 변수에 그 가지에 해당하는 값을 할당한다.

문제의 해를 찾는 과정은 제약조건을 만족하고 목적함수를 최소화하는 시작 노드에서 마지막 노드까지의 경로를 구하는 것이다. 어떤 노드의 가지가 해에 이르는지 미리 알 수 없기 때문에 root node에서 마지막 노드까지 모든 가능한 경로를 탐색해야 되는 경우도 있다. 하지만 실제문제에 적용할 경우 모든 가능한 경로의 탐색이 불가능하다. 1000개의 가능한 값을 갖는 10개의 제약변수의 경우는 탐색공간이 10^{1000}개에 이르게 된다. 이런 경우 해의 계산시간(computational time)이 몇 년이 걸릴 수도 있다. 복잡한 제약조건식 때문에 주어진 문제의 해를 찾는 것이 종종 어려울 때가 있다. 이것을 극복하는 한 가지 방법은 해를 찾기 위한 의사결정이 만들어질 때마다 남아 있는 의사결정들을 매우 빠른 속도로 추론하여 실행 불가능한 지역(infeasible area)을 줄여 나감으로써 탐색과정의 "combinatorial explosion"을 현저히 감소시킬 수 있다. 이러한 일련의 과정은 의사결정과정이 수행되고 있는 동안 동적으로(dynamically) 탐색공간을 줄이고 원하는 해로부터 현재의 탐색과정이 얼마나 떨어져 있는지를 추론한다.

ILOG Solver는 이러한 특징을 가지는 자체의 탐색알고리즘을 제공한다. 한 제약변수의 의사결정이 만들어질 때마다 나머지 변수들에 대한 영향을 추론하고 그 변수들의 도메인에서 실행 불가능해지는 대안(alternatives)을 제거하여 해를 찾는 데 걸리는 노력을 감소시킨다.

Domain reduction과정은 다음의 원리로 구현된다. 하나의 의사결정변수의 도메인을 줄였을 때, 이 의사결정변수에 대해 알려진 정보를 갱신한다. ILOG Solver는 이 정보를 이용해서 남아 있는 의사결정변수의 변경사항에 대한 새로운 정보를 추론한다. 위의 과정은 스케줄링의 예와 같이 설명될 수 있다. 예를 들어 한 작업의 가능한 시작시간을 줄인 결과가 7월 3일에서 5일 사이에 이 작업이 시작되어야 한다면 그 작업에 선행작업들은 반드시 7월 3일 전에 수행되어야 하고 그 작업에 후행하는 모든 작업들은 제일 빠른 시작시간이 7월 5일이라는 것을 추론할 수 있다.

2.2 ILOG Schedular

ILOG Schedule은 Activity 및 Activity에 의해 공유되는 자원측면에서 스케줄링 제약조건의 표현을 가능하게 하는 C++ 개발 라이브러리이다. ILOG Schedule은 사전에 정의된 다음과 같은 객체 지향적 제약조건 스케줄링 모델을 제공한다(ILOG, 1997d)

• 임시제약조건은 Activity 간의 선행관계를 표현한다.

• Resource Availability Constraint는 Activity가 자원을 어떻게 사용하는지, 공유하는지를 명시한다.

이런 제약조건은 다양한 스케줄링 문제에서 포괄적으로 적용된다. ILOG Schedule은 ILOG Solver에 추가되어 사용하는 제약조건문제 해결을 위한 C++ 라이브러리이다. 이 두 개의 툴을 사용한 응용해법의 개발은 다음의 방법을 따라야 한다.

1. 스케줄링 문제는 ILOG Schedule의 스케줄링 Object모델로서 표현된다.
2. 필요하다면 이런 표현은 ILOG Solver의 제약조건 정의 메커니즘을 이용해 특정 상황의 제약조건으로 확장될 수 있다.
3. 스케줄링 알고리즘은 Solver의 탐색 프로그래밍 기법을 이용해 구현된다.

3. 자원제약하의 동적 다중 프로젝트 일정계획

본 연구의 일정계획 시스템은 크게 4단계로 구성되어 있다. 1

단계는 프로젝트의 정보를 입력하는 단계이다. 여기에서는 발주된 프로젝트의 작업(job)과 자원(resource)에 관한 정보를 텍스트 파일로 작성하여 입력한다. 2단계는 입력받은 정보를 토대로 총처리시간을 최소로 하기 위한 일정계획의 수립단계이고, 3단계는 수립된 일정계획의 결과를 Gantt chart로 출력시켜주는 단계이다. 4단계는 동적으로 발생하는 문제, 즉 예를 들어 새로운 프로젝트의 수주를 처리할 수 있는 재일정계획단계로서 그후 2, 3단계를 반복하게 한다. <그림 1>은 일정계획 시스템의 구성과 정보의 흐름을 보여주고 있다.

시스템 구성 파일은 작업(job)에 대한 사용자 정의 class를 정의하는 Job.h,과 Gantt chart를 비롯한 GUI 환경을 구현하는 Main.cpp, 일정계획을 수립하는 Job.cpp, 작업과 자원에 대한 정보를 수록한 job.txt, resource.txt로 구성되며, 이 텍스트파일들을 입력받아 Main.cpp와 job.cpp를 분리 컴파일하여 시스템을 생성한다. 여기에 구성된 파일들의 자세한 code는 생략한다.

3.1 제1단계(정보입력단계)

자원제약하의 동적 다중 프로젝트 일정계획 시스템의 제 1 단계에서는 수주된 프로젝트의 자원과 작업에 대한 정보의 입

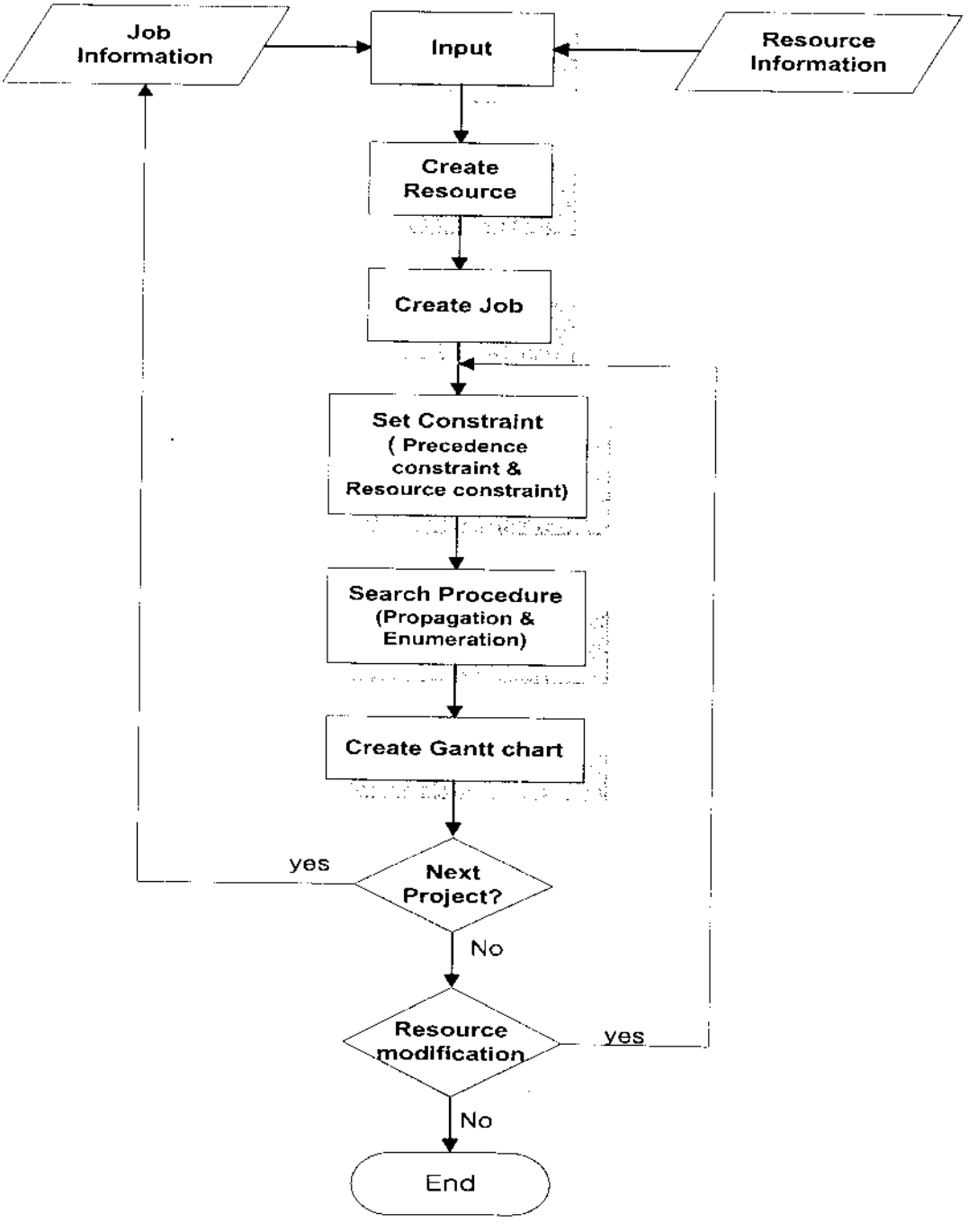

그림 1. 일정계획시스템의 구성과 정보의 흐름.

력단계이다. 자원과 작업에 관한 정보를 각각 resource.txt, job.txt라는 텍스트파일에 작성한다.

먼저 resource.txt에 작성한 정보를 살펴보면 먼저 맨 윗줄에 프로젝트에 이용되는 자원의 총 개수를 적고 다음 줄부터 한 줄씩 차례로 자원의 이름을 작성한다. 그 다음 job.txt에서는 첫 줄에 나오는 숫자는 작업의 개수를 나타내고 그 다음 첫줄부터 예로 들면 순서대로 프로젝트 색인, 선행노드,후행노드, 작업의 수행시간(duration), 작업이 이용하는 자원의 이름이 작성된다. 위와 같은 정보는 작업을 의미하는 객체 배열 Job들의 생성시 필요한 정보들이다.

3.2 제2단계(일정계획 수립단계)

가. 일정계획문제의 정의

본 연구에는 C++ 라이브러리인 ILOG Solver와 ILOG Schedule을 사용하였고, 일정계획 수립 프로그램은 Solving Schedule()이라는 하나의 서브루틴으로 작성하였으며 크게 일정계획문제를 정의하는 부분과 일정계획의 해를 찾는 부분으로 나누어진다.

ILOG Schedule에서는 IlcActivity class와 IlcUnaryResource class를 통해서 작업과 자원에 관련된 C++ 객체 생성을 할 수 있게 하고 있다.

1단계 자료 입력 단계에서 텍스트 파일로 입력된 프로젝트의 작업과 자원에 관한 정보를 이용하여 작업과 자원에 관한 객체를 생성한다. 전자는 createJobs(), 후자는 createResource()라는 함수를 이용한다.

함수 createJobs()안에서 작업에 관한 객체의 생성은 Job클래스의 생성자를 이용한다.

작업과 자원에 관한 객체를 생성한 후에는 setConstraint()함수를 이용하여 작업과 그 작업이 필요로 하는 자원과의 관계를 표현하는 resource constraint와 작업들 간의 선후행관계를 표현하는 precedence constraint를 생성한다.

다음 예는 ILOG Schedule에서 precedence constraint와 resource constraint를 표현하는 예이다.

- Activity2 . startsAfterEnd (Activity1)
  ; Activity2는 Activity1이 종료한 다음에 시작한다.
- Activity3 . requires Resource1
  ; Activity3는 Resource1을 이용한다.

일정계획 문제의 정의부분에서 일정계획의 최적기준(optimization criterion)을 프로젝트의 총처리시간을 최소화하는 것으로 선언한다.

나. 일정계획 알고리즘

일정계획문제의 정의가 끝나면 ILOG Solver에 의해 initial propagation이 발생하여 각각의 제약조건변수들의 도메인은 줄지만 bound(하나의 값으로 정해짐)되지는 않는다. 줄어든 도메인으로 원하는 일정계획의 해를 얻기 위한 알고리즘은 다음과 같이 구성한다.

절차1. : 순서(order)가 결정되지 않은 작업들이 이용하는 자원들 중 가장 " 결정적인 자원(critical resource) "을 선택한다.

절차2. : 선택된 자원을 이용하는 순서가 결정되지 않은 작업들 중에서 제일 먼저 수행할 작업을 선택하고 이에 해당하는 precedence constraint를 생성한다.

절차3 : 절차2를 선택된 자원에 해당하는 모든 작업들의 순서가 정해질 때까지 반복한다.

절차4 : 절차1에서 3까지 모든 자원에 해당하는 작업들의 순서가 결정될 때까지 반복한다.

절차 1에서 어떤 자원을 먼저 선택하는가 하는 의사결정은 일정계획에 있어 상당히 중요한 의미를 갖고 있다. 일반적으로 일정계획기간 동안 하나의 자원을 균등히 이용하지는 않는다. 즉 어떤 일정한 기간 동안 하나의 자원은 다른 자원들보다 더 많이 이용된다. 이 자원을 일정한 기간에 대해서 "결정적(critical)"이라고 한다. 이러한 자원의 제한된 유용성(availability) 때문에 프로젝트의 총처리시간을 줄이는 데 있어서 큰 어려움을 겪게 된다. "결정적 자원" 의 이용을 최적화하기 위해 "결정적 자원"을 먼저 선택하여 그 자원에 해당하는 작업을 일정계획해주는 것이 중요하다.

본 연구에서 "결정적 자원"을 선택하는 방법은 MinSlack 알고리즘을 이용하였다. 즉, 특정한 일정기간 동안 자원의 효용성(supply)과 자원에 대한 수요(demand)를 비교하여 최소의 여유(slack)를 갖는 자원을 선택한다. 여유에 대한 정의는 다음과 같다. 일정한 기간 동안 작업의 자원요구량(demand, 이 기간 동안 수행되어야 할 작업의 수행시간의 합)과 가용자원량(supply, 이 기간의 폭)의 차이이다.

다음의 예를 보면, 고려되는 일정한 기간의 시작은 순서를 정해야 하는 작업들 중의 가장 빠른 EST(Earliest Start Time)가 되고 일정한 기간의 끝은 가장 늦은 LFT(Latest Finish Time)가 된다. 자원의 수용력(capacity)은 1이므로 LFT에서 EST의 차이가 가용자원량(supply)이 되고 이 기간 동안 수행되어야 할 작업의 수행시간의 합이 자원요구량(demand)이 된다.

절차 2에서 선택된 자원을 이용하는 작업 중에서 제일 먼저 선택하는 작업은 가장 작은 EST를 갖는 작업을 선택한다. 만약 동일한 EST를 갖는 작업이 2개 이상 있다면 그 다음 조건으로 가장 작은 LST(Latest Start Time)를 갖는 작업을 먼저 선택한다.

위와 같은 절차 1에서 절차 4까지의 일정계획 알고리즘은 문제의 크기 및 해의 결과를 비교하면서 다른 유형의 알고리즘도 고려해 볼 수 있으며 현재 후속 연구가 진행중이다.

다. 해의 검색 (Solution search)

위의 과정이 끝난 후에 일정계획 알고리즘으로 생성된 precedence constraint에 의해 다시 제약조건변수의 도메인을 줄이고 ILOG Solver에서 제공하는 함수 nextSolution(), restart()를 이

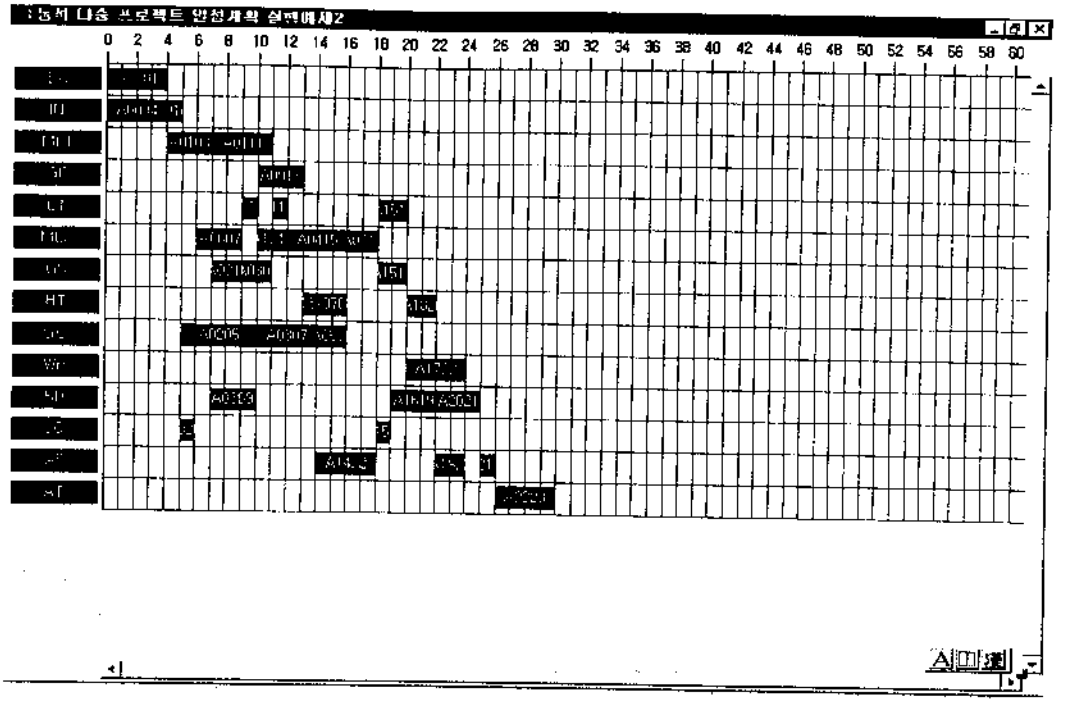

그림 2. 프로젝트 A의 일정계획 결과.

용하여 해의 검색(solution search)을 실시한다. 총처리시간을 최소로 하기 위한 해를 구하면 해의 출력을 보여주고 이 해를 Gantt chart로 출력하기 위해 일정계획 시스템의 메인 파일인 main.cpp로 리턴한다.

3.3 제3단계(일정계획 결과출력)

2단계에서 수립된 일정계획의 결과를 ILOG Views와 인터페이스하여 Gantt chart로 출력하고, 풍선 도움말 기능, 화면 확대, 축소 기능을 적용하여 사용자에게 편의를 제공한다.

3.4 제4단계(재일정계획 수립단계)

1단계에서 3단계를 거치는 동안 기존 프로젝트에 관한 일정계획이 수립되었다. 제4단계는 일정계획이 수립된 이후 동적으로 발생하는 새로운 프로젝트에 대한 문제에 대처하기 위한

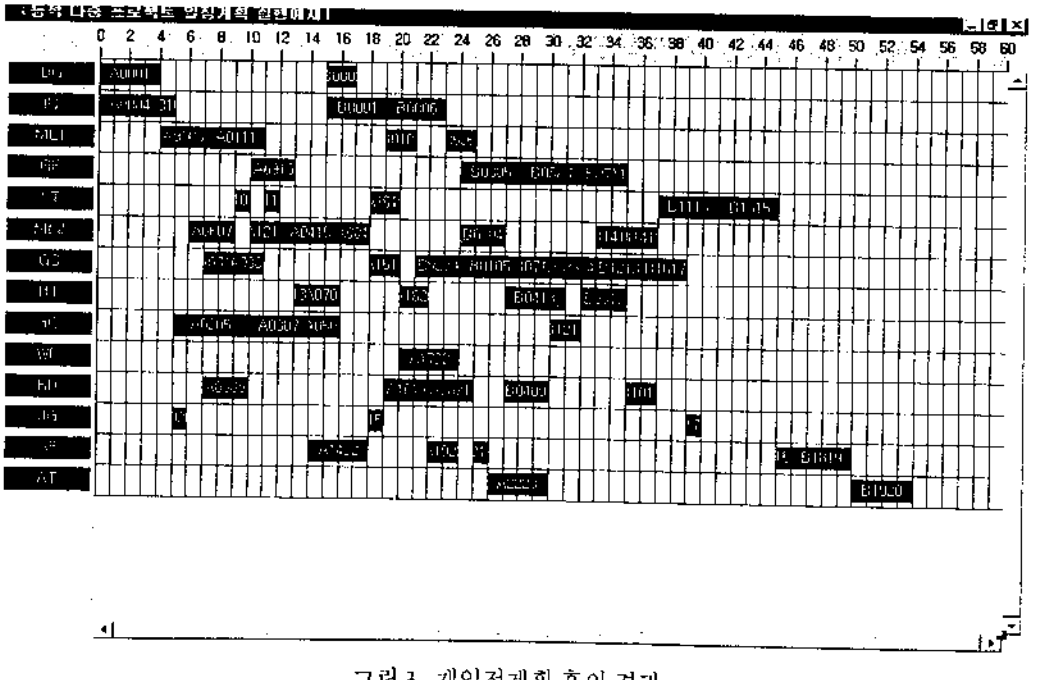

그림 3. 재일정계획 후의 결과.

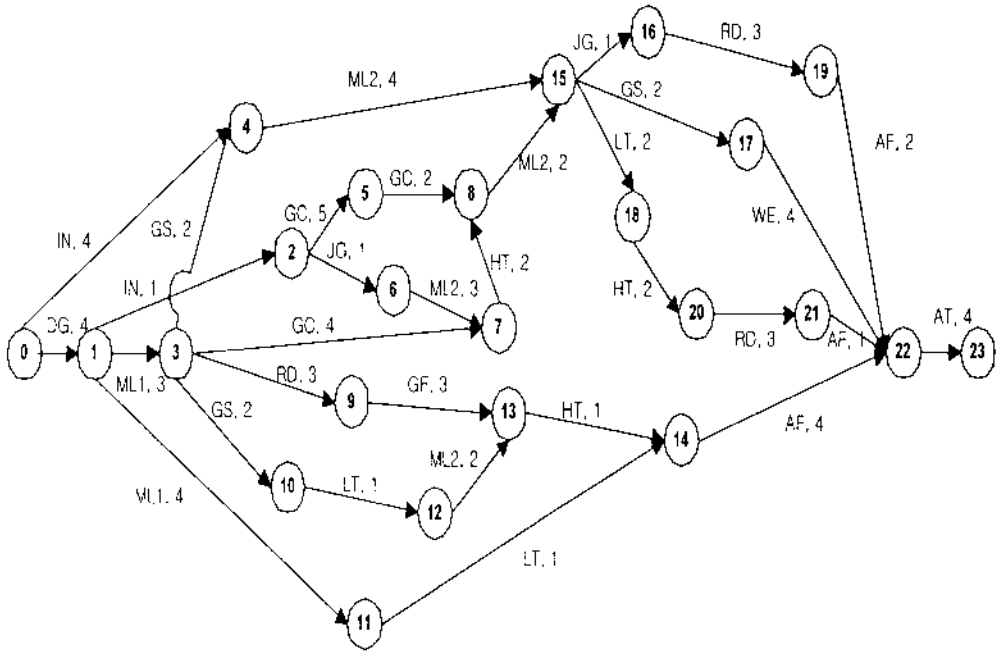

그림 4. 수치 예 1.

단계이다.

자원제약하의 동적 다중 프로젝트 일정계획 문제에서는 일정계획이 수립된 후 새로운 프로젝트의 수주로 인한 일정계획의 전면적인 수정이 요구된다. 이 경우 프로그램 알고리즘의 변경 없이 실시간으로 재컴파일하여 재일정계획을 실행한다.

<그림 2>는 프로젝트 A(4장 <그림 4>의 수치예 1)라고 명명한 기존 프로젝트에 대한 단계 1, 2와 3을 거친 일정계획 결과이다. 일정계획의 수립 후 프로젝트 B(4장 <그림 5>의 수치예 2)라고 명명된 또 다른 프로젝트가 15일 경과 후에 수주되었을 경우, 단계 1로 가서 작업에 관한 정보를 입력하는 기존의 텍스트파일 안에 첫 줄의 총작업의 수를 추가된 프로젝트의 수를 합한 총작업의 수로 변경하고 프로젝트 A의 입력줄 밑에 프로젝트 B의 작업의 정보를 입력한다. 또한 일정계획을 생성하는 파일인 job.cpp 안에 setStartMin()이라는 함수를 이용하여 프로젝트 B의 시작 시간을 15로 정한다. 또한 프로젝트 A의 15일 이전까지의 일정은 setStartTime()함수에 의해서 고정시킨다. 그 이후 프로그램은 단계 2와 3을 다시 수행한다.

다음의 <그림 3>은 프로젝트 B가 15일 경과 후 새로이 수주되었을 경우의 프로젝트 A와 같이 재일정된 결과이다. 이 경우 15일까지의 프로젝트 A의 일정에는 변경이 없지만, 15일 이후의 프로젝트 A의 작업들은 프로젝트 B의 작업들과 섞여 자원사용의 효율성에 따라 재일정된다.

4. 실행 예제

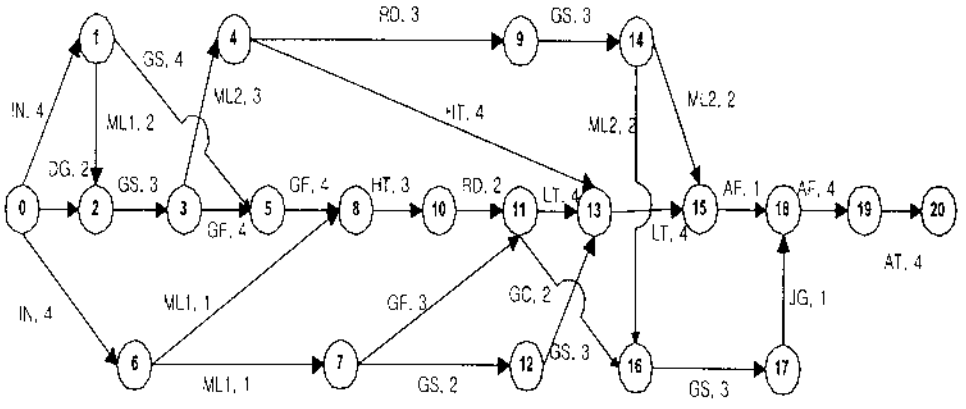

그림 5. 수치 예 2.

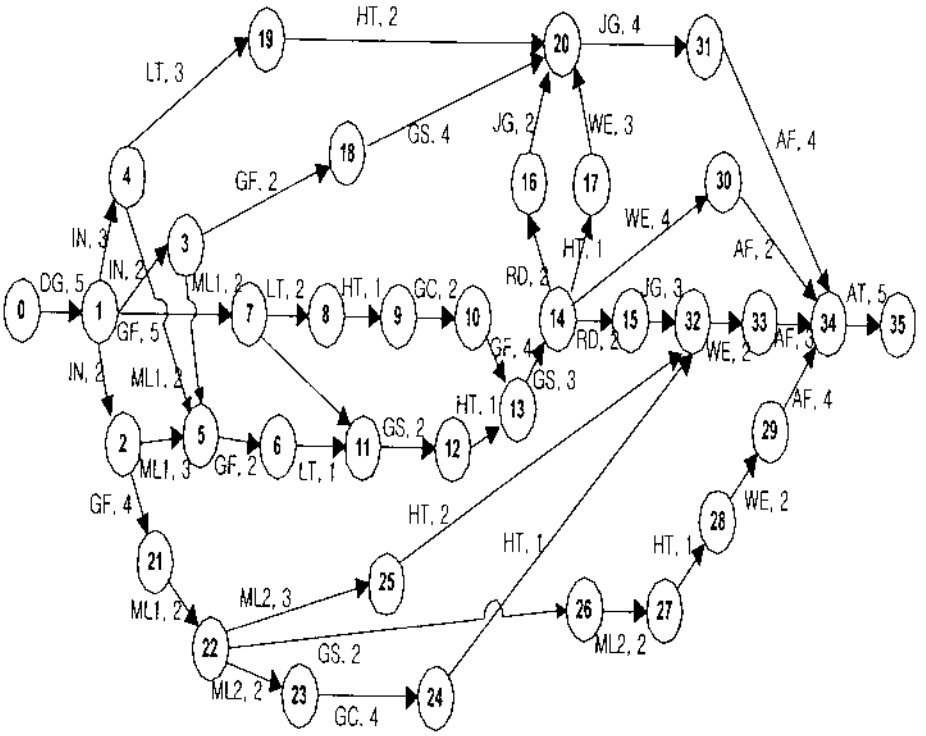

그림 6. 수치 예 3.

4.1 모의실험의 결과 비교

본 절에서는 여러 가지 형태의 프로젝트 예들을 제시하고, 앞 장에서 제시한 절차로 생성된 일정계획과 MINLFT 휴리스틱 규칙(이화기, 하승진, 1996)이나 TS기법(이화기, 윤종준, 1997)의 결과와 비교한다.

본 연구의 모의 실험에 사용된 프로젝트는 임의로 3개의 프로젝트들을 생성한다. <그림 4>의 프로젝트 네트워크를 수치 예 1, <그림 5>의 프로젝트 네트워크를 수치 예 2, <그림 6>의 프로젝트 네트워크를 수치 예 3으로 하여 모의 실험한다. 수치 예의 프로젝트 네트워크상에서 임의의 호에 표시된 영문 표기는 필요설비에 대한 약어이고 숫자는 작업시간을 의미한다. MINLFT 휴리스틱 규칙과 TS기법에 의한 일정계획과 본 연구에서 제시한 일정계획 절차를 모의 실험한 결과는 <표 1>과 같다.

<표 1>에서 실행 1은 먼저 수치 예 1이 수주된 후에 15일부터 20일까지의 시간 간격을 가지며 수치 예 2가 수주되었을 경우이고, 실행 2는 수치 예 1과 수치 예 3이 동적으로 수주되었을 경우에 대해 모의 실험한 결과이다. <표 1>에서 보여주듯이 실행 1은 모두 같은 결과를, 실행 2는 총처리시간(Makespan)에서 본 연구의 절차가 가장 우수함을 보여주고 있다.

4.2 금형가공 예

표 1. 총처리시간 비교

구 분	도착시간	실행 1	실행 2
MINLFT	15일	54	64
	17일	56	66
	20일	59	69
TS 기법적용	15일	54	62
	17일	56	64
	20일	59	67
본연구 절차적용	15일	54	60
	17일	56	62
	20일	59	67

모델은 현재 인천의 남동 공단에 위치한 금형 및 프레스 가공업체인 S사이다. 이 공장의 금형 가공에 사용되는 설비종류는 모두 12대이며, 작업을 수행하기 위해 사용되고 있는 실제 프로젝트 네트워크는 <그림 7>의 프로젝트 A와 <그림 8>의 프로젝트 B이다.

프로젝트 A가 수주된 후에 프로젝트 B가 15일 경과 후 수주되었을 경우, 본 연구에 의한 일정계획의 총처리시간은 94일로 계산된다. 각 작업들의 일정은 <표 2>와 같으며 그것의 Gantt chart 출력은 <그림 9>와 같다.

한편, MINLFT규칙에 의해 생성된 일정계획의 총처리시간

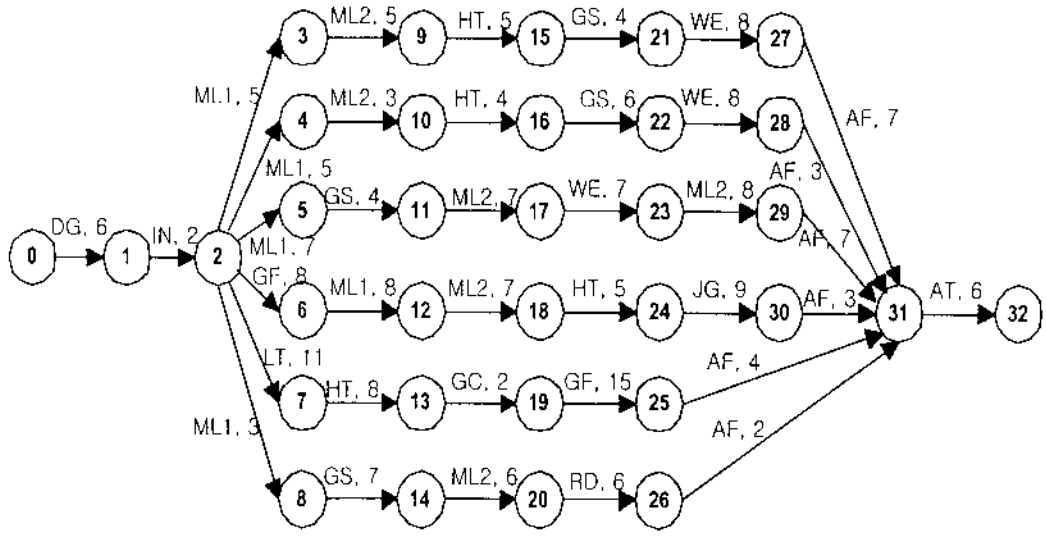

그림 7. 프로젝트 A.

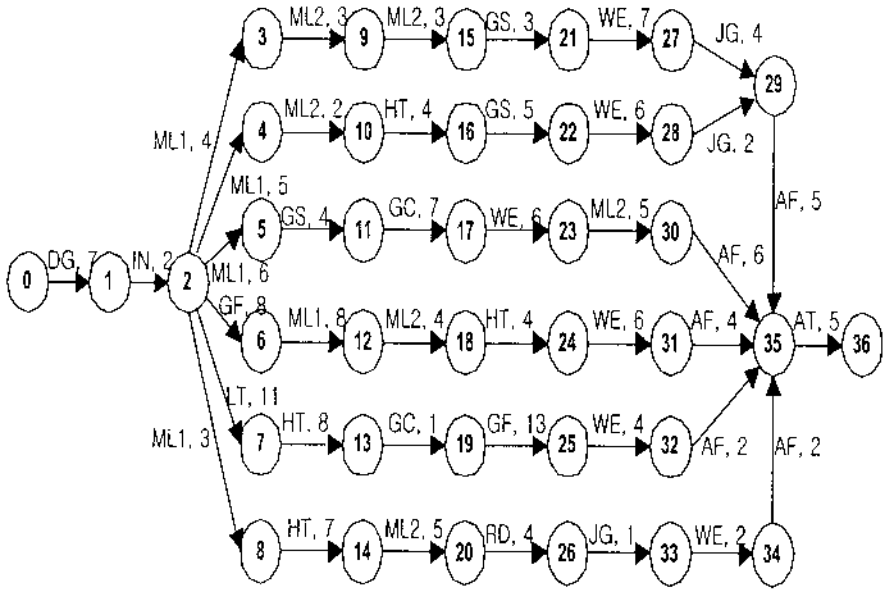

그림 8. 프로젝트 B.

은 117일이 되며, Tabu Search에 의한 결과는 102일로 나타났다.

4. 결 론

본 연구에서는 프로젝트 네트워크가 동적인 상황에서 다중으로 수주되었을 경우 생성되는 일정계획 문제에서 Constraint Programming을 기초로 한 C++ 라이브러리인 ILOG Schedule과 Solver에 Minslack 알고리즘을 적용하여 일정계획의 해를 구하였다.

다른 기법으로 제시된 해와 비교한 결과, 본 연구에서 제시된 일정계획 절차를 적용하였을 경우 기존의 휴리스틱 규칙보다 해의 개선을 보여주었다.

본 연구에서는 연구의 목적을 총처리시간(makespan)의 최소화에 두었는데 문제의 크기가 커지는 경우 바람직한 해를 구하기 위한 알고리즘의 계산시간(computation time)도 상당히 중요한 비중을 차지한다. 즉, 복잡한 제약조건식과 큰 탐색범위를 갖고 있는 문제의 적용시 알고리즘의 계산시간이 상당한 문제가 될 수 있다.

자원제약하의 동적 다중 프로젝트 일정계획 문제의 ILOG Solver와 Schedule의 이용시 다른 종류의 탐색 알고리즘이나 Tabu Search(TS), Simulated Annealing(SA)과 같은 메타휴리스틱의 접목을 통하여 프로젝트의 규모가 큰 문제를 다룰 수 있는 효율적인 일정계획의 개발에 대한 연구가 추후의 과제라고 할 수 있다.

표 2. 일정계획 결과

프로젝트 A					프로젝트 B				
Activity	Duration	Start Time	End Time	Resource	Activity	Duration	Start Time	End Time	Resource
(0 1)	6.0	0	6	DG	(0 1)	7.0	15	22	DG
(1 2)	2.0	6	8	IN	(1 2)	2.0	22	24	IN
(2 3)	5.0	11	16	ML1	(2 3)	4.0	37	41	ML1
(2 4)	5.0	46	51	ML1	(2 4)	5.0	41	46	ML1
(2 5)	7.0	16	23	ML1	(2 5)	6.0	31	37	ML1
(2 6)	8.0	8	16	GF	(2 6)	8.0	24	32	GF
(2 7)	11.0	8	19	LT	(2 7)	11.0	24	35	LT
(2 8)	3.0	8	11	ML1	(2 8)	3.0	59	62	ML1
(3 9)	5.0	16	21	ML2	(3 9)	3.0	41	44	ML2
(4 10)	3.0	53	56	ML2	(4 10)	2.0	47	49	ML2
(5 11)	4.0	23	27	GS	(5 11)	3.0	37	40	GS
(6 12)	8.0	23	31	ML1	(6 12)	8.0	51	59	ML1
(7 13)	8.0	26	34	HT	(7 13)	8.0	35	43	HT
(8 14)	7.0	11	18	GS	(8 14)	7.0	62	69	HT
(9 15)	5.0	21	26	HT	(9 15)	3.0	44	47	ML2
(10 16)	4.0	56	60	HT	(10 16)	4.0	49	53	ML2
(11 17)	7.0	27	34	ML2	(11 17)	6.0	40	46	GC
(12 18)	7.0	34	41	ML2	(12 18)	4.0	61	65	ML2
(13 19)	2.0	34	36	GC	(13 19)	1.0	46	47	GC
(14 20)	6.0	21	27	ML2	(14 20)	5.0	73	78	ML2
(15 21)	4.0	27	31	GS	(15 21)	3.0	47	50	GS
(16 22)	6.0	60	66	GS	(16 22)	5.0	53	58	GS
(17 23)	7.0	39	46	WE	(17 23)	6.0	46	52	WE
(18 24)	5.0	43	48	HT	(18 24)	4.0	69	73	HT
(19 25)	15.0	36	51	GF	(19 25)	13.0	51	64	GF
(20 26)	6.0	27	33	RD	(20 26)	4.0	78	82	RD
(21 27)	8.0	31	39	WE	(21 27)	7.0	52	59	WE
(22 28)	8.0	66	74	WE	(22 28)	6.0	59	65	WE
(23 29)	8.0	65	73	ML2	(23 30)	5.0	56	61	ML2
(24 30)	9.0	48	56	JG	(24 31)	6.0	74	80	WE
(25 31)	4.0	51	55	AF	(25 32)	4.0	80	84	WE
(26 31)	2.0	33	35	AF	(26 33)	1.0	82	83	JG
(27 31)	7.0	39	46	AF	(27 29)	4.0	59	63	JG
(28 31)	3.0	78	81	AF	(28 29)	2.0	65	67	JG
(29 31)	7.0	73	78	AF	(29 35)	5.0	67	72	AF
(30 31)	3.0	56	59	AF	(30 35)	6.0	61	67	AF
(31 32)	6.0	81	87	AT	(31 35)	4.0	81	85	AF
					(32 35)	2.0	85	87	AF
					(33 34)	2.0	84	86	WE
					(34 35)	2.0	87	89	AF
					(35 36)	5.0	89	94	AT

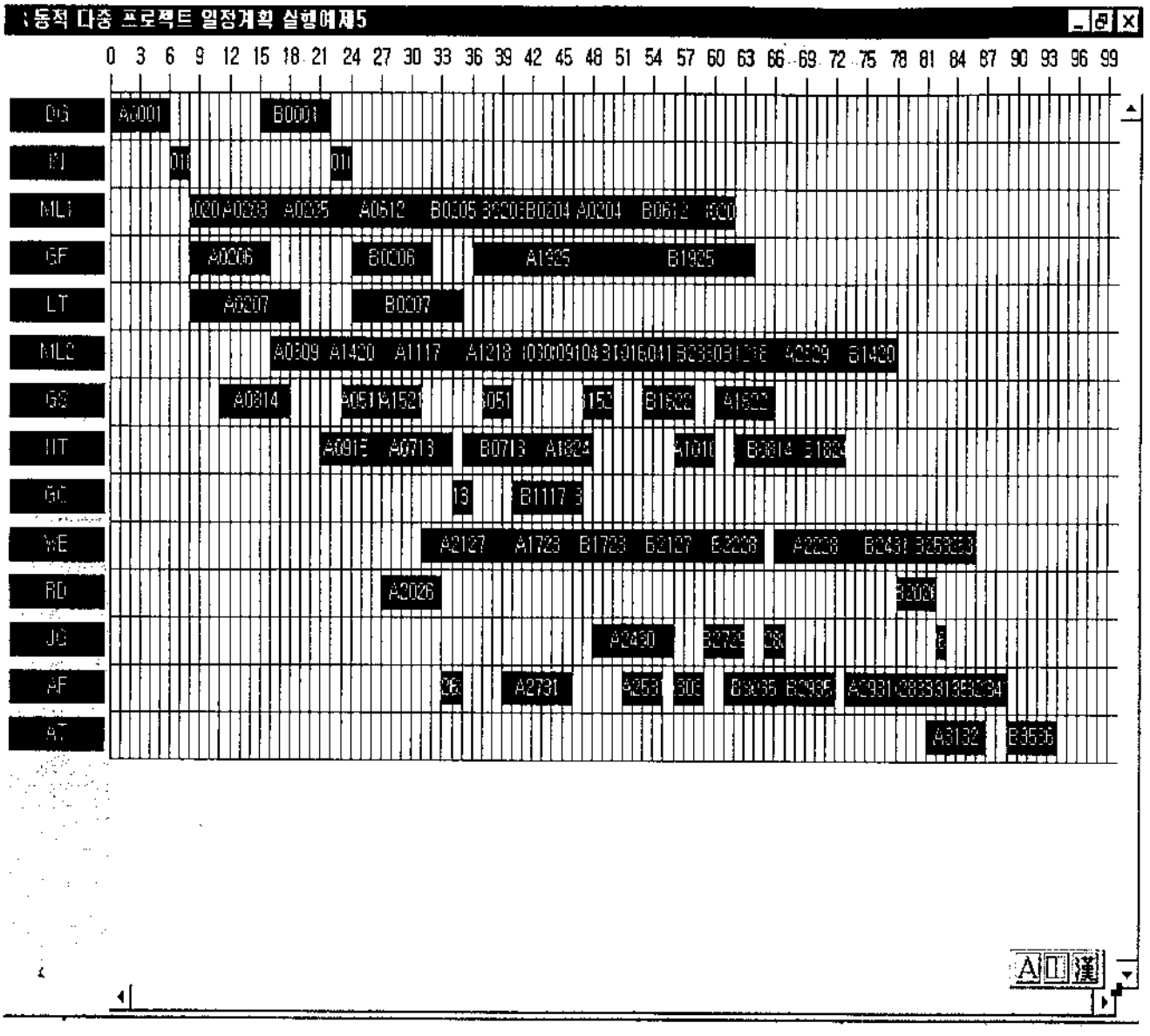

그림 9. 일정계획 결과 Gantt Chart.

참고문헌

박성현 (1976), Scheduling theory and problem : review and categorization of solution procedure, *대한산업공학회지*, 2(1), 101-108.

이화기, 하승진 (1996), 자원제약하의 동적 다중 프로젝트 일정계획에 관한 연구, *인하대학교 산업기술연구소 논문집*, 24(1), 89-97.

이화기, 윤종준 (1997), Tabu Search를 이용한 자원제약하의 동적 다중 프로젝트 일정계획, *인하대학교 산업기술연구소 논문집*, 26, 411-426.

Kurtulus, I. and David, E. W. (1982), Multi-project scheduling: categorization of heuristic rules performance, *Management Science*, 28(2), 161-172.

Dumond, J. and Marbent, V. A. (1988), Evaluating projects scheduling and due date assigment procedure: an experiment analysis, *Management Science*, 34(1), 101-118.

Claude, L. P. (1994), Constraint-based programming for scheduling historical perspective, *ILOG S.A. Working Paper*, Operations Research Society Seminar on Constraint Handling Techniques, London.

Douglas, R. S. and Stephen, J. W. (1995), Synthesis of constraint algorithms. *Principles and Practice of Constraint Programming*, The MIT Press, 173-182.

ILOG (1997a), ILOG Schedule User Manual 4.0, ILOG.

ILOG (1997b), ILOG Schedule White Paper, ILOG.

ILOG (1997c), ILOG Solver User Manual 4.0, ILOG.

ILOG (1997d), ILOG Solver White Paper, ILOG.

ILOG (1997e), ILOG Views User Manual 2.4, ILOG.

아화기

1979 서울대학교 원자핵공학과 학사
1981 Texas A&M University 산업공학과 석사
1984 Texas A&M University 산업공학과 박사
현재: 인하대학교 산업공학과 교수
관심 분야: 생산시스템의 최적화, Simulation

정제원

1996 인하대학교 산업공학과 학사
1998 인하대학교 산업공학과 석사
관심 분야: Scheduling