▩ 응용논문

CSP와 SA를 이용한 Job Shop 일정계획에 관한 연구

-A Study on the Job Shop Scheduling Using CSP and SA-

윤 종 준*
Yun, Joung Jun
손 정 수*
Son, Jeong Su
이 화 기**
Lee, Hwa Ki

Abstract

Job Shop Problem which consists of the m different machines and n jobs is a NP-hard problem of the combinatorial optimization. Each job consists of a chain of operations, each of which needs to be processed during an uninterrupted time period of a given length on a given machine. Each machine can process at most one operation at a time. The purpose of this paper is to develop the heuristic method to solve large scale scheduling problem using Constraint Satisfaction Problem method and Simulated Annealing.

The proposed heuristic method consists of the search algorithm and optimization algorithm. The search algorithm is to find the solution in the solution space using CSP concept such as backtracking and domain reduction. The optimization algorithm is to search the optimal solution using SA.

This method is applied to MT06, MT10 and MT20 Job Shop Problem, and compared with other heuristic method.

1. 서론

Job Shop Problem은 m대의 기계를 이용하여 서로 다른 가공시간과 가공순서를 가진 n개의 job을 일정계획하는 문제로 NP-hard로 분류되는 조합 최적화 문제이다. 본 논문의 목적은 Job Shop Problem을 대상으로 대규모 일정계획문제에 대한 효과적인 일정계획 시스템을 개발하는 것이다. 이를 위해 최근 관심을 받고 있는 인공지능기법의 하나인 CSP(Constraint Satisfaction Problem)기법과 메타 휴리스틱 기법의 하나인 SA(Simulated Annealing)을 사용하여 발견적 기법을 제시하고자 한다.

본 연구에서는 CSP의 적용을 위해 ILOG라는 C++ 프로그래밍 라이브러리를 이용하고 있다. 그러나 문제의 크기가 커질 경우 해를 구하기 위한 효율적인 방법이 요구되는데, 이를 위해 일반적으로 사용되는 메타 휴리스틱의 하나인 SA(Simulated Annealing)을 사용하여 해의 탐색 범위를 결정하고 있다. 제안된 발견적 기법의 알고리즘은 크게 해를 탐색해 나가는 과정과 이 과정에서 얻어진 해의 개선을 통해 최적해를 찾기 위한 최적화 과정으로 구분된다. 해의 탐색

* 인하대학교 대학원 산업공학과
** 인하대학교 산업공학과

과정에서는 Backtracking 탐색을 이용하여 해를 찾는 것이며, 최적화 과정에서는 탐색 과정에서 고려되는 해의 범위를 SA을 이용하여 보다 나은 해를 찾을 가능성을 높이는 것이다.

본 연구의 결과를 평가하기 위해 JSP의 벤치마킹 문제로 널리 알려진 MT06, MT10, MT20 문제에 적용하고 얻어진 해를 기존의 연구결과와 해의 총 처리시간(makespan) 및 계산시간(CPU time) 면에서 본 연구와 비교한다.

2. 이론적 고찰

2.1 Constraint Satisfaction Problem(CSP)

CSP 방법은 인공지능(AI) 방법의 일종으로써 제약조건을 만족하는 해를 찾아나가는 방법을 가리킨다. 즉, 문제를 구성하는 제약조건(C)과 유한 이산 도메인(D)을 가지는 제약변수(V)를 가지고 문제를 모델링 한 후 여러 가지 인공지능(AI)기법이나 OR기법들을 동원하여 제약조건을 만족하는 제약변수의 값을 구하고자 하는 방법이다.

기본적으로 CSP를 해결하는 기법은 모든 가능해 집합을 나열하여 가능해나 최적해를 찾아내는 것이다. 단점은 해를 찾아내는데 걸리는 검색시간이 오래 걸린다는 점이다. 따라서 CSP 기법의 핵심은 어떻게 검색시간을 줄일 수 있느냐는 점에 있다. CSP문제에 주로 적용되는 검색기법으로는 Backtracking을 들 수 있다[9][10][11].

CSP기법을 이용한 시스템의 기본적인 특징은 첫째, 제약식의 정의부분과 Constraint Propagation 부분, 문제해결을 위한 검색알고리즘 부분을 서로 분리하고 있다. 둘째, 각 제약식은 가능한 한 국부적으로 Propagation되도록 한다. 즉, 선언부에서 선언된 제약식 만을 가지고 불필요한 Domain을 Propagation을 통해 줄인 다음, OR이나 AI기법들을 이용한 적절한 검색알고리즘을 이용하여 해를 구하게 된다[9][10].

2.2 Simulated Annealing (SA)

조합적 최적화 문제의 한 해법으로 Kirkpatrick et al(1983)에 의해 제안된 SA는 기존의 반복적인 개선(Iterative Improvement)에 근거한 발견적 기법(Heuristic Methods)들이 지역 최소점(Local Minimum Point)에 빠져버리는 단점을 개선한 범용의 최적화 기법으로 현재 많은 분야에서 응용되고 있다[1].

SA는 고체물리학에서 에너지 수준이 가장 낮은 상태인 결정을 얻기 위해서 이용하고 있는 어닐링 과정을 그대로 최적화 문제에 적용하는 것이었다. 어닐링을 확률적으로 묘사하는 문제는 통계학분야에서 1953년 개발된 Metropolis 알고리즘을 사용한다. Metropolis 알고리즘은 열탕에서 고체가 열 평형 상태에 도달하는 과정의 확률적 시뮬레이티드 방법으로 과정은 다음과 같다. 고체의 초기상태를 무작위적으로 선택한 후 현재 상태 X에서 약간의 변동(perturbation)을 주어 다음의 상태 Y를 만들고 에너지의 차이 $\triangle E = E(Y)-E(X)$가 0이하이면 Y로의 변동을 받아들이고 0보다 크면 Y로의 변동을 확률 $\exp(-\triangle E/kBT)$로 받아 들여(이것을 Metropolis 기준이라고 함) 현재 상태를 변경시키는 과정을 반복한다. 온도를 고정시키고 이 과정을 충분히 많이 반복하면 결국은 고체의 상태 분포가 볼쯔만 분포를 따르게 되어 열 평형에 도달하게 된다[1][3][6].

SA는 Metropolis 알고리즘을 기본으로 하여 확률적으로 비용의 증가를 허용하여 반복적 개선에 의해 지역 최소점에 빠지는 단점을 해결하고 전체 최소점을 찾아간다[23]. SA는 아직까지 적용 상에서 해결되지 않은 문제점들을 가지고 있으며, 특히 계산 시간이 오래 걸린다는 단점을 가지고 있다. 따라서 해의 질을 떨어뜨리지 않고 계산 시간을 줄이기 위해 SA의 변형을 이용한 발견적 기법을 고려할 필요가 있다[1].

2.3 ILOG

프랑스 ILOG사에서 개발된 ILOG제품군은 통신, 제조, 운송 등의 분야에서 발생하는 자원제약 문제를 해결하기 위한 제품으로써 문제를 해결하는 부분인 Optimization제품군(ILOG Solver, Scheduler 등)과 이를 사용자와 인터페이스를 위한 Visualization제품군(ILOG Views 등)으로 이루어져있다. 이들 제품들은 C++의 객체지향설계의 개념에 의해 만들어졌기 때문에 코드의 재 사용성 및 새로운 프로그램 개발에 걸리는 시간의 단축, 다른 응용프로그램과의 연결 등 확장성에 있어서도 이점을 갖고 있으며 많은 사례 연구에 적용되고 있는 제품이다.

ILOG Solver는 통신, 제조, 운송 등의 분야에서 CSP를 해결하기에 적합한 고수준의 프로그래밍 C++라이브러리로서 미리 정의된 변수 클래스, 새로운 제약과 결합이 가능한 제약 클래스, 새로운 발견적 기법을 적용할 수 있는 탐색 알고리즘으로 이루어져 있다. 이처럼 클래스를 통해 OOP개념을 실천함으로써 코드의 재활용 및 확장성이 용이하게 되었으며 실제 발생하는 여러 가지 상황에도 적용될 수 있다[7][9].

ILOG Schedule은 Solver에 추가되어 제약조건문제의 스케줄링 문제와 자원할당문제를 해결하기 위한 C++라이브러리이다. 기본적으로 Solver에서 제공하는 변수와 제약조건에 관한 Class를 이용하며 Activity와 Resource라는 개념을 사용하여 문제를 표현하고 이를 해결하는 역할을 한다[8].

3. CSP와 SA를 이용한 일정계획

3.1 ILOG와 사용자 정의 알고리즘의 결합

ILOG는 CSP에 기반을 둔 Constraint Programming C++ 라이브러리로 구성되어 있다. Constraint Programming은 문제를 구성하고 있는 변수와 제약조건을 선언하는 과정, 제약조건에 만족하는 변수들의 값을 찾아나가는 해의 탐색과정으로 구분할 수 있다. 실제로 Solver를 사용할 때에는 해의 탐색과정에서 문제에 따라 Solver가 제공하는 알고리즘을 사용하거나, 복잡한 문제를 해결하기 위해서 Goal Programming이라는 새로운 사용자 정의의 알고리즘을 사용할 수 있도록 하고 있다[4][9][10].

본 연구에서는 탐색과정에서 해를 찾는데 실패할 경우 그전의 상태로 복귀하는 Backtrack의 수를 이용하여 탐색을 진행하고 일정 횟수의 반복이 발생할 때 알고리즘을 종료하는 것으로 했다. 그리고 좀더 효과적인 탐색에 의한 최적해를 찾기 위해, SA의 특징 중 하나인 비용의 증가를 확률적(Metropolis 기준)으로 허용하여 다음단계에서 보다 개선된 해의 탐색가능성을 높였다.

3.2 CSP와 SA를 이용한 발견적 기법의 제시

3.2.1 Parameter 구성

본 연구에서 사용하는 Backtrack(해의 탐색과정에서 변수의 도메인을 찾기에 실패한 변수로부터 그전 결정 상태로 되돌리는 과정)수와 반복 수는 일정횟수의 Backtrack이 발생할 때 하나의 반복과정을 마치는 것으로 하거나 정해진 Backtrack 내에서 해가 찾아지면 그 해를 이전의 해와 비교하여 개선이 되었을 때 받아들이고 반복 수를 증가시킨다. 이 과정에서 어느 정도의 Backtrack을 사용할 것인가에 따라 해가 발견될 확률과 계산시간에 영향을 미친다. 따라서 Backtrack수를 고정시키는 방법보다는 비용함수에 따라 SA의 확률적 판단 기준(Metropolis 기준)을 사용하여 변화를 주어 계산시간에는 별 영향을 주지 않으면서 해를 개선하는 것이 본 연구가 제시하는 발견적 기법이다.

3.2.2 비용함수의 구성

본 연구에서 SA적용에 필요한 비용함수는 해의 총 처리시간(makespan)의 도메인 영역을 이용하는데 방법은 다음과 같다. 즉, 총 처리시간이 그림 1(a)와 같은 도메인을 갖는다면 그 사이에서 해가 구해질 것이다. 비용함수는 그 도메인 사이에서 임의의 값(a)을 선정하여 해가 구해질 도메인 영역을 그림 1(b)와 같이 선정하고 구간 A에서 해를 구한다. 따라서 새로운 제약조건을 만드는 것이다. 만약 해를 구하는데 실패할 경우는 그림 1(c)와 같이 도메인 영역에서 다시 임의의 값(b)을 설정하고 구간 B에서 해를 구하게 된다. 비용함수를 이용하여 도메인의 영역이 작을수록 좀더 많은 수의 Backtrack을 하도록 하여 해를 찾아갈 확률을 높이고자 했다.

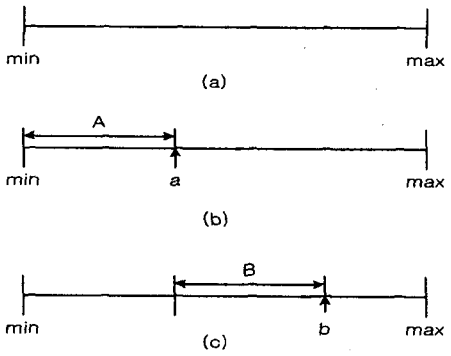

그림 1. 비용함수(총 처리시간)의 도메인 영역변화

3.3 알고리즘 구성

3.3.1 탐색 알고리즘

해를 찾아나가는 탐색 알고리즘은 CSP를 이용하여 제약조건을 만족하는 해를 전체 가능해 공간에서 찾아나가는 과정이다. 이는 각 제약변수가 제약식을 만족하는 도메인 값을 갖게 되는 것을 의미한다. 이 과정에서 어떤 변수를 먼저 선택하여 해를 구할 것인가를 해결하기 위해 기존의 연구에서는 Local Slack이나 Global Slack 등을 이용한 Minslack 방법, 또는 문제에 적절한 발견적 방법들을 개발하여 사용한다. 본 연구에서 제시하는 탐색 알고리즘의 단계는 다음과 같으며 탐색 알고리즘의 흐름도는 그림 2와 같다.

<탐색 알고리즘 단계>

단계 1 : 일정계획 되지 않은 자원의 모든 Activities 중에 처음으로 일정계획이 될 수 있는 것들을 검사한다.

단계 2 : 검사된 자원 중에서 Activities의 수가 가장 적은 자원(Ci)을 선택하여 먼저 일정계획을 실시한다.

단계 3 : 모든 자원의 일정계획이 될 때까지 단계 1과 2를 반복한다.

3.3.2 최적화 알고리즘

SA 알고리즘 적용부분으로 찾아진 해의 개선과 탐색에 있어 개선된 해를 찾기 위한 최적화 알고리즘의 단계는 다음과 같으며 최적화 알고리즘의 흐름도는 그림 3과 같다.

<최적화 알고리즘 단계>

단계 1 : 초기화 단계에서는 알고리즘을 진행하는 파라메터를 초기화한다.

단계 2 : 초기해를 생성한다. 초기해는 제약조건의 Propagation 만을 사용하여 구하며 총 처

리시간의 도메인을 구하게 된다.

단계 3: 찾아진 해의 총 처리시간의 도메인을 이용한 비용함수를 이용하여 탐색에 필요한 Backtrack 수를 결정한다. 이때 비용함수의 확률적 판단(Metropolis 기준)에 의해서 좀더 많은 수의 Backtrack 수를 설정하거나 초기치를 이용한다.

단계 4 : 결정된 Backtrack수를 이용하여 탐색 알고리즘을 적용하고 해를 구한다. 해를 구하는데 실패한다면 반복 수를 증가시키며 단계 3으로 이동한다.

단계 5 : 구해진 해와 비교하여 현재 해를 갱신하고 단계 3으로 이동한다.

단계 6 : 정해진 수의 반복이 끝나면 알고리즘을 종료한다.

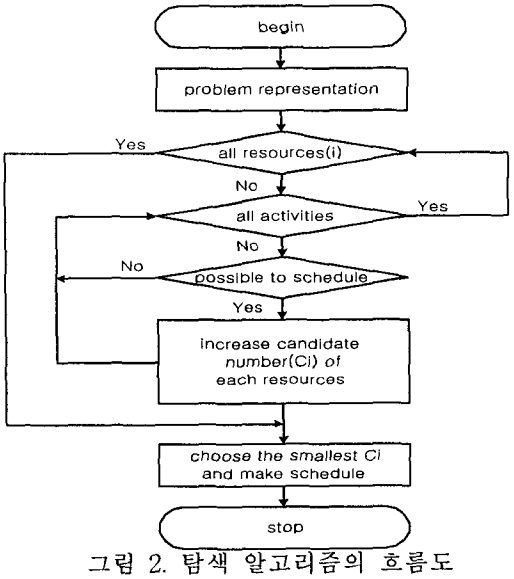

그림 2. 탐색 알고리즘의 흐름도

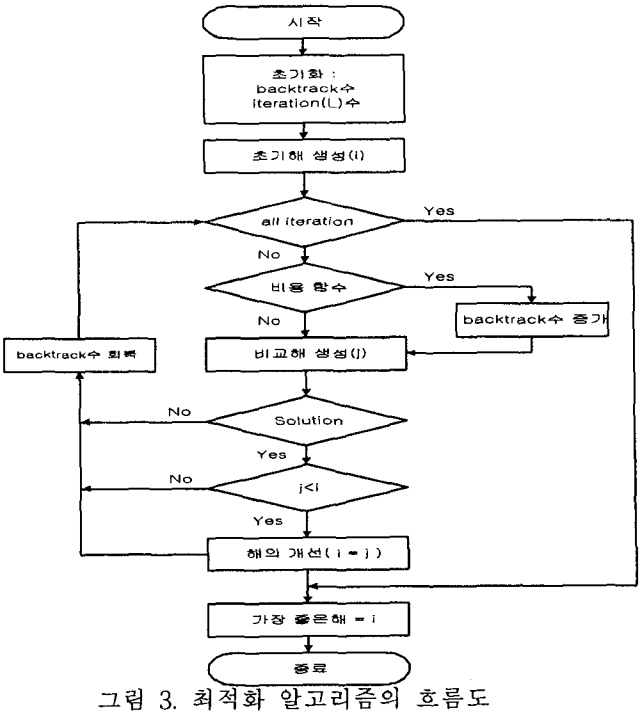

그림 3. 최적화 알고리즘의 흐름도

4. Job shop 문제의 적용 예

4.1 비교분석

JSP에서 목적함수가 총 처리시간의 최소화인 문제에 대해 각각 실험을 통하여 본 연구에서 제안된 발견적 기법과 기존 연구를 비교분석 하고자 한다.

실험에 사용된 문제는 MT06(6×6), MT10(10×10), MT20(20×5)으로 알려진 벤치마크(benchmak)문제를 사용하였다. 이 문제들은 1963년 최초로 제안된 이후 JSP에 관한 많은 연구에서 비교분석 문제로 사용되어 왔다. MT06 문제는 6개의 작업들과 6개의 기계를, MT10 문제는 10개의 작업들과 10개의 기계를, MT20 문제는 20개의 작업들과 5개의 기계를 이용하여 일정계획하는 문제이며 기존의 연구를 통해 이미 최적해를 알고 있는 문제이다[14]. 실험에 사용된 컴퓨터는 CPU P-133이며 계산시간은 CPU시간이다.

4.2 실험 방법 및 결과

실험에 필요한 입력 파라메터에 따라 각각의 실험을 실시하였다. 우선 확률적 판단 기준이 되는 비용함수에 필요한 파라메터(SA에서는 온도설정)를 경험적 방법에 의해 4가지로 구분했다. 이들은 임의로 발생하는 0에서 1사이의 확률 값과 비교되어 Backtrack 수를 늘릴 것인지를 결정하는 파라메터로서 값이 클수록 1에 가까워져서 거의 모든 경우에 받아들이게 되며 이는 계산시간의 증가를 가져오게 된다. 따라서 적정한 확률을 발생시킬 수 있는 값이 요구된다. 본 연구에서는 표 1의 4가지의 경우로 구분하여 각각 실험했다. 또한 알고리즘의 효율성 및 속도에 영향을 주는 Backtrack 수는 표 2의 5가지 경우로 구분하여 각각 실험하였고, 실험에서 반복 수는 100회로 하였다.

표 1. 온도의 구분

구분	A1	A2	A3	A4
온도 값	1000	900	800	700

표 2. backtrack 수의 구분

구분	B1	B2	B3	B4	B5
초기 backtrack수	10	10	20	20	50
증가된 backtrack수	30	50	30	50	100

이를 기준으로 MT06, MT10, MT20의 실험을 실시한 결과는 표 3, 4, 5와 같고, 이를 gantt chart로 나타낸 그림은 그림 4, 5, 6이다.

표 3. MT06 실험 결과

MT06(opt = 55)		비용함수에 필요한 파라메터			
		A1	A2	A3	A4
B1	총 처리시간	55	55	55	55
	계산시간	0.44	0.72	0.50	0.60
B2	총 처리시간	55	55	55	55
	계산시간	0.60	0.60	0.50	0.55
B3	총 처리시간	55	55	55	55
	계산시간	0.60	0.61	0.55	0.60
B4	총 처리시간	55	55	55	55
	계산시간	0.61	0.55	0.72	0.77
B5	총 처리시간	55	55	55	55
	계산시간	0.60	0.66	0.55	0.77

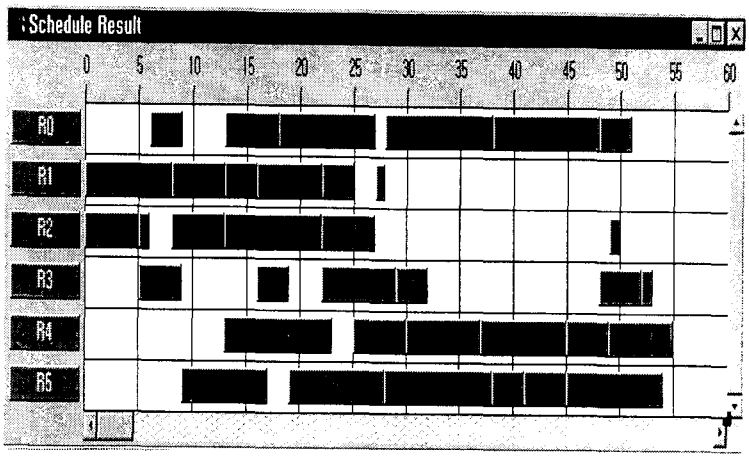

그림 4. MT06 gantt chart

표 4. MT10 실험 결과

MT10(opt = 930)		비용함수에 필요한 파라메터			
		A1	A2	A3	A4
B1	총처리시간	993	993	993	993
	계산시간	81.02	80.68	80.25	78.65
B2	총처리시간	993	993	993	993
	계산시간	107.38	108.59	105.18	103.59
B3	총처리시간	993	993	993	993
	계산시간	86.51	87.71	86.17	86.62
B4	총처리시간	993	993	993	993
	계산시간	113.26	113.75	112.32	110.62
B5	총처리시간	988	988	988	988
	계산시간	107.89	106.86	106.25	105.32

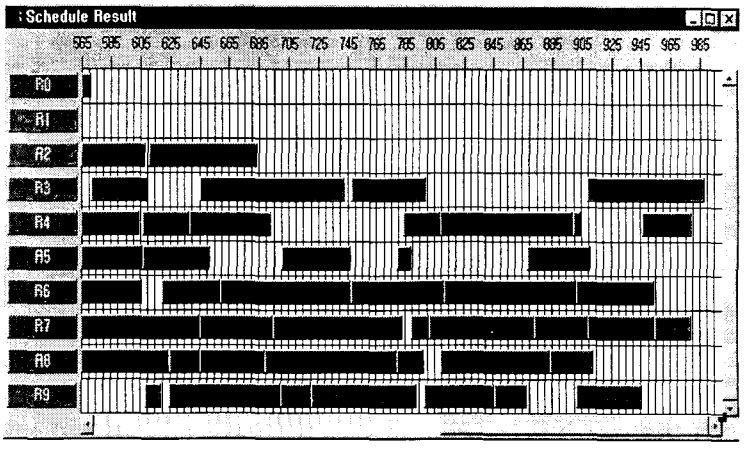

그림 5. MT10 gantt chart

표 5. MT20 실험 결과

MT20(opt = 1165)		비용함수에 필요한 파라메터			
		A1	A2	A3	A4
B1	총처리시간	1165	1165	1165	1165
	계산시간	50.04	55.53	49.43	50.21
B2	총처리시간	1165	1165	1165	1165
	계산시간	54.15	53.33	53.12	52.40
B3	총처리시간	1165	1165	1165	1165
	계산시간	49.55	49.38	50.09	50.70
B4	총처리시간	1165	1165	1165	1165
	계산시간	50.80	48.89	49.93	48.72
B5	총처리시간	1165	1165	1165	1165
	계산시간	38.33	38.34	38.83	38.61

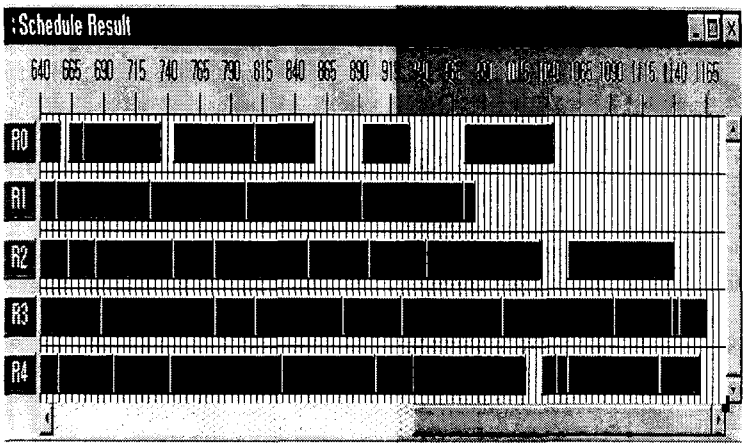

그림 6. MT20 gantt chart

연구에서 발견적 기법을 적용한 실험결과 MT06, MT20은 빠른 시간 내에 최적해를 찾았다. 그러나 MT10의 경우 최적해는 구하지 못하였다.

4.3 기존 연구와의 비교

얻어진 결과를 기존의 연구와 총 처리시간(makespan) 및 계산시간(CPU time)면에서 비교하였다. 우선 기존의 연구와 총 처리시간 면에서 비교하면 다음 표 6과 같다.

표 6. 기존연구와 본 알고리즘의 비교

기존 연구 (최적해)	MT06 (55)	MT10 (930)	MT20 (1165)
Genetic Algorithm 적용[5]	55	938	1178
Tabu Search 기법적용[2]	55	966	1180
제안된 본 알고리즘	55	988	1165

비교 결과 MT10에서는 Genetic Algorithm이나 Tabu Search 적용 연구가 좋은 결과를 MT20에서는 본 연구가 좋은 결과를 얻었다. 계산시간은 사용된 컴퓨터의 차이로 무의미하다. 표 7은 본 연구가 제시하는 발견적 기법과 Randomizing Algorithm[8]의 비교 결과이다.

표 7. Randomizing Algorithm과 본 알고리즘의 비교

문제	Opt	Randomizing algorithm[8]		본 알고리즘	
		총 처리 시간	계산시간	총 처리 시간	계산시간
MT06	55	55	0.61	55	0.44
MT10	930	996	54.48	988	194.82
MT20	1165	1165	67.67	1165	38.33

비교 결과 MT06, MT20 문제에서는 월등한 수행시간 내에 최적해를 구하였다. 그러나 MT10 문제의 경우 총 처리시간 면에서 약간의 개선은 있었지만 수행시간에서는 오히려 증가하는 결과를 얻었다. 이는 제시한 알고리즘에서 확률적으로 Backtrack 수를 증가시킴으로써 보다 많은 수의 탐색이 일어나게 한 결과이다.

5. 결론

본 연구는 일정계획문제가 점점 대규모화되는 현실에서 적절한 시간 내에 적절한 해를 구하기 위한 발견적 기법에 관한 것이다. 이를 위해 최근 사용되는 CSP를 기반으로 하는 ILOG Programming Library를 이용했고, CSP가 가지는 문제점인 전체 탐색공간에 대한 탐색으로 인한 시간적인 문제점을 해결하기 위해 일반적인 개선알고리즘으로 받아들여지는 SA를 이용했다. 즉, 제안된 CSP와 SA를 이용한 일정계획 기법은 탐색 알고리즘 부분과 해의 개선을 위한 최적화 알고리즘 부분으로 구성되어있다. 최적화 알고리즘 단계에서 SA의 기본 개념인 확률적으로 나쁜 이동을 받아들이는 방법과, CSP기법의 backtrack수를 파라메터로 이용하여 적절한 시간 내에 문제를 해결할 수 있도록 했다.

실험 결과 제시된 발견적 기법을 JSP 벤치마크 문제의 총 처리시간의 최소화 문제에 적용하여 기존 연구의 발견적 방법보다 해의 향상이나 시간 면에서 좋은 결과를 얻을 수 있었다. 추후 연구과제로는 본 연구에서 제시된 알고리즘을 실제 대규모 JSP 일정계획 또는 자원할당 문제에의 적용이라 할 수 있다.

참고 문헌

[1] 김여근, 윤복식, 이상복, *메타 휴리스틱*, 영지문화사, 1997.

[2] 김여근, 배상윤, 이덕성, "Job Shop 일정계획을 위한 Tabu Search", *대한산업공학회지*, Vol 21, No 3, 1995.

[3] Catoni, O., "Solving scheduling problems by simulated annealing", *SIAM J. Control Optim*, Vol 36, No 5, 1998.

[4] Desai, R and R. Patil, "Combining SA and local Optimization for Efficient Global Optimization", *In Proceedings of the 9th Florida AI Research Symposium*, 1996.

[5] Dorndorf, U. and Pesch, E. "Evolution based Learning in a Job Shop Scheduling Environment", *Computers & Operations Research*, 22(1) : 15-24, 1995.

[6] Golden, B.L and C.C. Skiscim, "Using Simulated Annealing to Solve Routing and Location Problems", *Naval Research Logistics Quarterly*, Vol 33, 1986.

[7] ILOG, *ILOG Solver manual*, ILOG, 1997.

[8] ILOG, *ILOG Scheduler manual*, ILOG, 1997.

[9] ILOG, "Optimization technology white paper", *ILOG, 1998.*

[10] ILOG, "ILOG Solver White Paper", *ILOG 1998.*

[11] Le Pape, C., "Constrant Based Programming for Scheduling : An Historical Perspective", *Operations Research Society Seminar on Constraint Handling Techniques*, Working Paper, 1994.