Journal of the Society of Korea Industrial and Systems Engineering
Vol. 25, No. 2, pp.92-101, April 2002.

Tabu Search와 Constraint Satisfaction Technique를 이용한 Job Shop 일정계획

윤종준 · 이화기

인하대학교 산업공학과

Job Shop Scheduling by Tabu Search Combined with Constraint Satisfaction Technique

Joung Jun Yun · Hwa Ki Lee

Dept. of Industrial Engineering, University of Inha

The Job Shop Scheduling Problem(JSSP) is concerned with schedule of m different machines and n jobs where 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. The purpose of this paper is to develop the efficient heuristic method for solving the minimum makespan problem of the large scale job shop scheduling.

The proposed heuristic method is based on a Tabu Search(TS) and on a Constraint Satisfaction Technique(CST). In this paper, ILOG libraries is used to embody the job shop model, and a CST is developed for this model to generate the initial solution. Then, TS is employed to overcome the increased search time of CST on the increased problem size and to refine the next-current solution. Also, this paper presents the new way of finding neighbourhood solution using TS. On applying TS, a new way of finding neighbourhood solution is presented.

Computational experiments on well known sets of MT and LA problem instances show that, in several cases, our approach yields better results than the other heuristic procedures discussed in literature.

Keywords : Job Shop Scheduling, Tabu Search, Constraint Satisfaction Technique.

1. 서론

본 연구에서 다루고자 하는 Job Shop Scheduling Problem(JSSP)는 m대의 기계를 이용하여 서로 다른 가공시간과 가공순서를 가진 n개의 job을 일정계획 하는 문제이다. JSSP는 잘 알려져 있는 것처럼 일종의 조합최적화 문제이면서 NP-hard 문제이다[8]. 따라서 문제가 가지고 있는 복잡성과 넓은 탐색영역 때문에 수리계획법, 분지 한계법 등과 같은 수리적 방법으로는 문제의 크기가 작을 경우 최적 해를 구할 수 있지만, 문제의 크기가 커질 경우 계산 량의 증가로 적절한 계산시간 내에 최적해에 근사한 해를 찾아낼 수 없다.

본 연구의 목적은 문제를 이루고있는 작업과 기계자원이 커지는 대규모 JSSP를 해결하기 위해서 Tabu Search(TS)와 Constraint Satisfaction Technique(CST)를 결합한 발견적 기법을 개발하고자한다.

연구에서 제안하는 발견적 기법은 CST를 기반으로 하는 프로그래밍 라이브러리인 ILOG(ILOG Solver 4.4,

Scheduler 4.4)를 이용하여 제약기반 근사 알고리즘을 구현한다. 대규모 JSSP의 경우 문제가 복잡하여 계산량이 증가할 경우 CST의 기본 탐색 방법으로는 문제해결이 불가능한 경우도 발생한다. 따라서 문제해결을 위해 또 다른 발견적 기법과의 결합이 필요하며, 이를 위해 연구에서는 최근 여러 분야에서 쓰이는 메타 휴리스틱인 TS를 적용하여 탐색의 한계를 극복하고 대규모 Job Shop 일정계획 문제를 해결할 수 있는 발견적 기법을 제안한다.

본 연구에서 제안된 발견적 기법의 객관적인 수행도 평가를 위해 JSSP 벤치마킹문제로 널리 알려진 MT문제 (Fisher 와 Thompson에 의해 제시, FT문제라고도 함)와 LA문제(Lowrence에 의해 제시) 중 몇몇 문제에 적용하여 얻어진 해를 기존의 연구결과와 해의 총처리시간(makespan) 및 계산시간(CPU time)면에서 본 연구와 비교한다.

2. 이론적 고찰

2.1 제약만족 기법(Constraint Satisfaction Technique : CST)

제약만족 기법(Constraint Satisfaction Technique : CST)개념은 인공지능(Artificial Intelligence : AI) 분야의 일종으로써, 주어진 제약조건을 만족하는 가능해를 찾는 것을 목적으로 하는 문제인 제약만족문제(Constraint Satisfaction Problem : CSP)의 해를 구하는 데 사용되는 방법의 총칭이다[2]. 즉, 제약만족 기법은 문제를 구성하는 제약조건과 제약변수를 가지고 문제를 모델링한 후 여러 가지 인공지능(AI)기법이나 OR기법들을 동원하여 제약조건을 만족하는 제약변수의 값을 구하고자 하는 방법이다. 여기에서 일정계획 문제는 문제가 가지고 있는 변수와 제약을 이용하여 제약만족문제로 표현된다.

CSP의 구성은 변수 $V=\{v_1, v_2, v_3, \cdots, v_n\}$, 변수가 가질 수 있는 도메인 $D=\{d_1, d_2, d_3, \cdots, d_n\}$ 그리고 이러한 변수들간의 제약조건 으로 이루어 $C=\{c_1, c_2, c_3, \cdots, c_n\}$ 진다. (여기서, n=변수의 수)[4]. CSP는 변수 V_n 에 대해 각 변수에 들어갈 값이 도메인 D_n 이라고 하면, 지수의 비율인 $O(V_n!)^{D_n}$ 의 계산 량을 갖기 때문에 변수나 도메인이 큰 경우에는 많은 시간이 소요된다. 이러한 문제에서 각 변수를 전향검사(Forward Checking)하여 제약조건의 일치성(Consistency)을 검사하고 이에 따라 도메인 여과(Domain Reduction)기법을 적용하여 백트랙킹(Backtracking)을 할 경우에는 전체 탐색 공간의 크기를 줄일 수 있다.

제약전파(Constraint Propagation)란 문제를 풀기 위하여 고려중인 변수의 범위(Domain)내에서 변수와 관련되는 제약식을 통하여 문제의 해에 포함될 수 없는 변수의 범위(변수의 값)를 삭제하는 도메인 여과 기법(Domain Reduction Technique)을 말한다.

연구에서 사용하는 ILOG제품군은 CSP를 기반으로 하는 프로그래밍 라이브러리로 통신, 제조, 운송 등의 분야에서 발생하는 자원제약 문제를 해결하기 위한 제품으로써 문제를 해결하는 부분인 Optimization 제품군(ILOG Solver, Scheduler 등)과 이를 사용자와 인터페이스를 위한 Visualization 제품군(ILOG Views)으로 이루어져 있다.

ILOG Solver는 CSP를 해결하기에 적합한 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 OOP 개념을 실천함으로써 코드의 재활용 및 확장성이 용이하게 되었으며 실제 발생하는 여러 가지 상황에도 적용될 수 있다[10].

ILOG Scheduler는 Solver에 추가되어 제약조건문제의 스케줄링 문제와 자원할당문제를 해결하기 위한 C++라이브러리이다. 기본적으로 Solver에서 제공하는 변수와 제약조건에 관한 클래스를 이용하며 Activity와 Resource라는 개념을 사용하여 문제를 표현하고 이를 해결하는 역할을 한다[11].

2.2 Tabu Search

TS는 어려운 조합최적화 문제를 해결하기 위한 새로운 AI 기법으로 근원은 1960년 대 말 1970년대 초로 거슬러 올라간다. TS의 기본적인 개념은 Hansen에 의해서 구상되었고, 몇 년후 Glover에 의해서 현재 형태의 TS가 제안되었으며[9], 외판원 문제, Flow Shop 문제, Job Shop 문제, Quadratic Assignment Problem(QAP) 등의 조합 최적화 문제에 효율적으로 적용되고 있다. 일반적으로 TS기법에서는 현재해에서 여러 이웃해를 생성함으로써, 해 공간의 탐색을 강화(intensification)하거나 다양하게 탐색(diversification)하기 위하여 최근에 생성된 한정된 해의 이동 정보를 타부목록(tabu list)에 기억하는 단기기억(short term memory)과 해의 이동 정보를 처음부터 현재까지 기억하는 장기기억(long term memory)을 이용한다. 또한 TS에서 고려되는 여러 이웃해를 생성하는 이동방법으로 해의 원소를 교환, 삽입, 삭제, 첨가하는 방법을 사용한다.

TS에서 Tabu 파라메타로는 해의 이동(move), 타부속성(tabu attributes), 타부목록(tabu list), 타부목록의 크기(tabu list size), 열망기준(aspiration criteria), 종료조

건(stop criterion) 등이다. TS는 이러한 파라메타를 문제의 특성에 맞게 조정함으로써 최적해나 근사해를 찾는 탐색시간을 줄일 수 있다. 해의 이동이란 초기해로부터 생성한 모든 이웃해 중에서 하나의 이웃해로 이동하는 것이고, 타부목록은 해의 이동정보를 일정기간(또는 일정횟수의 이동) 기억하는 것으로 해의 이동에 관한 정보를 나타내는 타부속성을 정의하여야 한다. 타부목록의 크기는 타부목록에 저장되는 타부속성의 개수를 나타낸다. 열망기준이란 어떤 이동이 타부상태에 있으나 해가 어느 기준 이상이면 이를 해제하여 해의 이동을 허락하는 기준이다.

TS를 실제문제에 적용하기 위해서는 문제의 특성에 맞는 Tabu 파라메타를 설정해야한다.

3. Job Shop 일정계획 문제해결을 위한 발견적 기법

3.1 탐색전략

본 연구의 목적은 JSSP의 총처리시간을 최소화하는 것이다. 제안된 발견적 기법은 두 가지의 잘 알려진 AI 기법인 TS와 CST을 이용하며, TS를 적용함에 있어 Tabu 파라메타가 탐색의 효율성을 증가시킬 수 있도록 고안하여 적용한다. 또한 탐색과정에서 해를 찾는데 실패할 경우 이전의 상태로 복귀하는 Backtrack 수를 이용하여 탐색을 진행하고, 정해진 일정횟수의 반복이 종료되거나 일정횟수동안 해의 개선이 없을 경우 알고리즘을 종료하도록 하였다.

또한 탐색 효율을 증가시키기 위해 ILOG[11]에서 CST를 적용하여 문제해결을 위해 제시한 Dichotomizing binary search 알고리즘에서 총처리시간 변수의 도메인 영역변화 기법을 본 연구의 이웃해 탐색과정에 이용한다. 이 절차는 발생한 총처리시간의 변수 값을 이등분하는 이진탐색을 한다.

제안된 발견적 기법에서는 우선 문제를 구성하고 있는 제약변수와 제약조건을 정의하고 각 제약변수가 제약조건을 만족하는 값의 영역(domain)을 찾아나가게 된다. 이 과정에서 기존의 연구에서는 어떤 변수(기계 또는 자원)를 먼저 선택하여 해를 구할 것인가를 해결하기 위해, Local slack, Global slack을 이용한 Minslack 방법[11], 또는 문제에 적절한 발견적 방법들을 개발하여 사용한다.

본 연구에서는 초기해 생성을 위한 부분에서 Minslack 방법을 사용하고, 초기해에서 이웃해 생성을 통한 해의 개선부분에서 TS 절차를 적용하면서, CST 기법의 제약식 추가와 제거를 통한 탐색방법의 발견적 기법을 제시하고 이용한다.

3.2 초기해 생성 알고리즘

CST에서 계산시간과 탐색회수를 감소시키기 위한 방법으로 총처리시간 변수의 초기 도메인 값을 정해주고, 제약전파를 해줌으로서 각 공정 변수들의 도메인을 어느 정도 정해준다. 이와 같이 변수들의 도메인 영역을 감소 한 후에 Minslack 알고리즘을 적용하여 초기해를 생성한다.

< 초기해 생성 알고리즘 >

단계 1 : 선행, 자원 제약 조건식의 제약전파만 이용하여 각 공정변수들의 도메인을 구한다.

단계 2 : Minslack 알고리즘을 적용한다. 즉, 순서(order)가 결정되지 않은 공정(activity)들이 이용하는 자원들 중 가장 "결정적인 자원(critical resource)"을 선택한다.

단계 3 : 선택된 자원을 이용하는 순서가 결정되지 않은 공정들 중에서 제일 먼저 수행할 공정을 선택하고 이에 해당하는 precedence constraint를 생성 한다.

단계 4 : 단계 2를 선택된 자원에 해당하는 모든 공정들의 순서가 정해질 때까지 반복한다.

단계 5 : 단계 1에서 3까지 모든 자원에 해당하는 공정들의 순서가 결정될 때까지 반복한다.

연구에서는 "결정적 자원"을 선택하는 방법으로 Minslack 알고리즘을 이용한다. Minslack 알고리즘은 특정한 일정기간동안 자원의 효용성(supply)과 자원에 대한 수요(demand)를 비교하여 최소의 여유(slack)를 갖는 자원을 선택한다. 여유(slack)란 일정한 기간동안 작업의 자원 요구량(demand : 이 기간동안 수행되어야 할 작업의 수행시간의 합)과 가용 자원량(supply : 이 기간의 폭(interval))의 차이이다.

일례로 <표 3-1>의 4×4 Job Shop 문제인 작업이 4개이고, 공정이 16개이며, 기계자원이 4개로 이루어진 문

<표 3-1> 4×4 Job Shop 문제

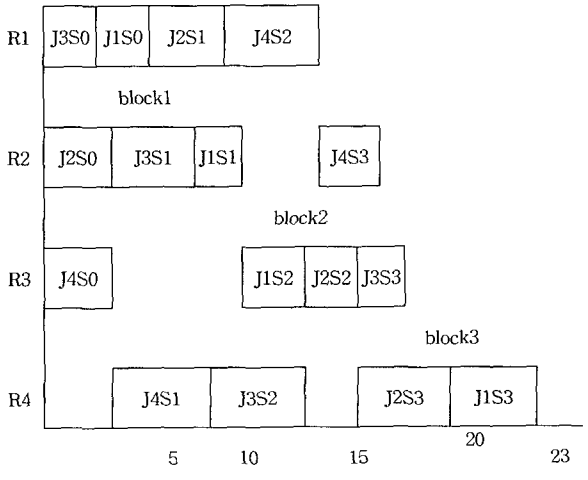
작업	공정순서(가공시간)			
1	1(2)	2(3)	3(2)	4(4)
2	2(3)	1(5)	3(3)	4(4)
3	1(2)	2(4)	4(4)	3(2)
4	3(3)	4(5)	1(4)	2(3)

<표 3-2> 4×4 Job Shop 문제

J1S0R1[2..5--2-->4..7]
J1S1R2[7--3-->10]
J1S2R3[10--2-->12]
J1S3R4[19--4-->23]
J2S0R2[0--3-->3]
J2S1R1[4..7--5-->9..12]
J2S2R3[12--3-->15]
J2S3R4[15--4-->19]
J3S0R1[0..1--2-->2..3]
J3S1R2[3--4-->7]
J3S2R4[8..11--4-->12..15]
J3S3R3[15..21--2-->17..23]
J4S0R3[0..3--3-->3..6]
J4S1R4[3..6--5-->8..11]
J4S2R1[9..16--4-->13..20]
J4S3R2[13..20--3-->16..23]

제가 주어졌을 때, 초기해 생성 알고리즘을 적용하여 생성된 해는 <표 3-2>와 같이 각 공정들이 가질 수 있는 도메인의 값으로 표현된다.

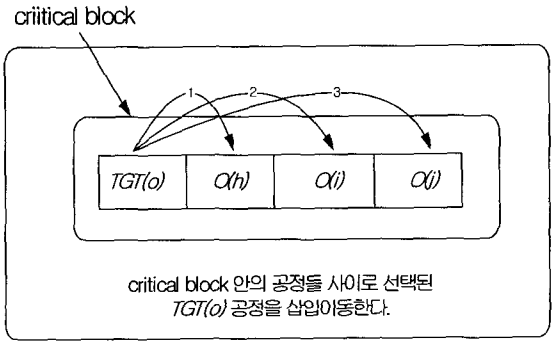
<표 3-2>의 해의 표현에서 각 공정의 이름이 주어지고(예 "J1S0R1" J는 작업, S는 순서, R은 이용자원) 정보사항은 괄호([ ])안에 표현된다. 정보는 세가지 항목인 각 공정의 시작시간, 가공시간, 완료시간이다. 각 항목은 단일 값 또는 구간 값으로 주어지며, 구간 값은 점 2개(..)로 표현된다. 예로 J1S1R2[7--3-->10]의 의미는 공정 J1S1R2은 7시점에서 시작하여 10시점에서 완료됨을 의미하고, J1S0R1[2..5--2-->4..7]의 의미는 공정 J1S0R1의 시작은 2와 5시점 사이에서 시작하여 공정시간 2단위시간 후에 4와 7시점사이에서 완료됨을 나타낸다. 여기서 공정 J1S0R1는 가장이른 시작시간부터 가장 늦은 완료시간사이에서 진행됨을 의미하나 전체 일정의 총처리시간에는 영향을 미치진 않는다. 그러나 J1S1R2 공정은 총처리시간에 영향을 미치는 critical 공정이다. <그림 3-1>은 얻어진 <표 3-2>의 해에서 각 공정의 가장이른 시작시간과 가장이른 완료시간 값으로 표현한 Gantt Chart이다. 그림에서 음영으로 표시된 공정은 총처리시간에 영향을 미치는 공정들로 critical 공정이라 정의하고, 이러한 공정들이 이루는 path가 critical path가 된다. 그림에서 critical path는 J2S0, J3S1, J1S1, J1S2, J2S2, J2S3, J1S3 공정들로 이루어진다. critical block이란 기계자원별로 이러한 critical 공정들이 이루어진 block을 말한다(block1, 2, 3).



<그림 3-1> 4×4문제의 초기해 Gantt Chart

3.3 이웃해 생성 알고리즘

총처리시간의 감소를 위해서는 Disjunctive Graph[21]의 critical path내의 공정간의 순서 변화를 통해서 이루어진다는 개념 하에, 연구에서는 critical path를 이루는 critical 공정들의 순서 변화를 통해서 알고리즘을 진행하며, 초기해에서 이웃해의 생성을 위해 모든 critical block을 고려하는 것이 아니라 critical 공정을 가장 많이 점유하고 있는 critical block과 기계자원을 선택한다. 선택된 critical block에서 삽입이동을 위해 공정을 선택하는 기준은 각 공정의 직전 선행공정의 시작시간(EST)과 직 후행공정의 완료시간(LCT)을 구하여 시간간격이 가장 큰 공정을 선택한다. 여기서 직전 선행공정과 직 후행공정이란 작업을 이루고 있는 공정들이 가공순서상에서 선후관계를 이루고 있는 공정을 말한다. 선택된 공정을 $TGT(O)$이라 정의하고 선택된 block 안의 모든 공정들 사이의 위치로 삽입 이동한다. <그림 3-2>은 선택된 공정 $TGT(O)$가 block안의 공정들 사이로 삽입이



<그림 3-2> $TGT(O)$공정의 삽입이동에 의한 이웃해 생성

동되어 이웃해를 생성하는 그림을 보여준다.

초기해에서 이웃해 생성을 위해 적용된 알고리즘 단계는 아래와 같다. 알고리즘 흐름에서 해의 효율적인 탐색을 위해 local search 기법인 one-step look-back interchange 절차[13]와 총처리시간 영역변화기법을 적용한다.

< 이웃해 생성 알고리즘 >

단계 1 : 초기해에서 critical path를 구하고, 자원별로 이루어진 critical block에서 가장 많은 수의 critical 공정들이 포함되어있는 critical block을 선택한다.

단계 2 : 선택된 critical block에서 각 공정의 직선행공정의 시작시간과 직후행공정의 완료시간을 구하여 시간간격이 가장 큰 공정을 선택하고, $TGT(O)$ 으로 둔다.

단계 3 : $TGT(O)$ 공정과 동일자원을 점유하는 critical block안의 공정들 사이로 순차적으로 삽입이동 한다. 이때 선택된 공정간의 삽입이동 관계를 제약조건으로 추가한다.

단계 4 : one-step look-back interchange 절차를 따른다.

절차 1 : 단계 3에서 삽입이동 한 후 삽입하는 위치에 있는 공정과 동일 작업의 직전 선행공정이 있는지 검사한다. 있으면 직전 선행공정을 선택한다. 이때의 자원도 선택한다.

절차 2 : 선택된 직전 선행공정의 선행 공정을 선택한다(동일한 자원을 공유). 선행공정이 있으면 직전 선행공정과 교환이동 한다. 이때 선택된 공정간의 교환이동 관계를 제약조건으로 추가한다.

단계 5 : 제약전파에 의해 하나의 이웃해를 구한다. 이때 해의 탐색시간을 줄이기 위해 일정횟수의 Backtrack 수를 설정하고, 총처리시간 변수의 영역변화 기법을 적용한다. 생성된 하나의 이웃해에 대해서 타부상태 검사 알고리즘을 적용한다.

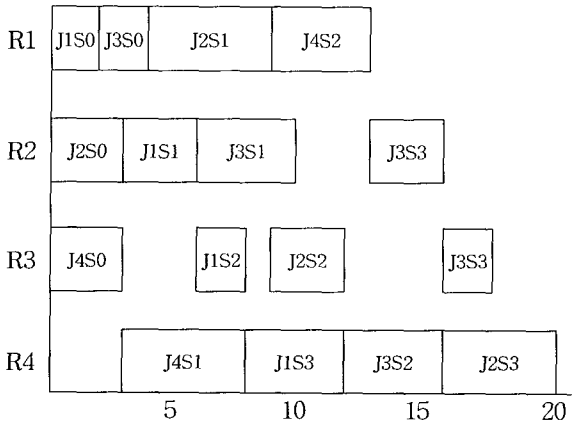
단계 6 : 단계 3에 의해 모든 이웃해를 생성하고, 얻어진 해 중에서 가장 좋은 해를, 최선해와 다음 Iteration 의 초기해로 설정한다.

이웃해 생성 알고리즘 적용 예를 <표 3-1>의 예제를 통해 설명하면, <그림 3-1>의 Gantt Chart에서 critical block은 block1(J2S0, J3S1, J1S1), block2(J1S2, J2S2), block3(J2S3, J1S3)가 된다. 알고리즘 흐름단계에서 이웃해 생성을 위해 선택된 critical block은 critical 공정이 가장 많은 block1이 되며, 삽입이동을 위해 선택되는 공정은 block1의 critical 공정들 중 선택공정의 직전 선행공정과 직 후행공정의 시간간격이 가장 큰 공정인 J3S1 공정이 선택된다. 선택된 공정이 block1의 공정들 사이로 삽입이동에 의해 생성되는 이웃해는 두 개로 block1의 공정순서는 이웃해 1(J3S1, J2S0, J1S1) 과 이웃해 2(J2S0, J1S1, J3S1)가 된다.

<표 3-3>은 이웃해 2의 공정순서로 이웃해 생성알고리즘을 적용하여 생성된 하나의 이웃해를 표현한 표이고, <그림 3-3>은 Gantt Chart로 나타낸 그림이다. Gantt Chart에서 음영으로 표시된 공정은 critical 공정이다. <그림 3-3>에서는 <그림 3-1>의 초기해의 총처리시간이 23에서 총처리시간이 20으로 줄어든 것을 볼 수 있다.

<표 3-3> 4×4 Job Shop 문제의 이웃해

J1S0R1[0..1--2-->2..3]
J1S1R2[3--3-->6]
J1S2R3[6--2-->8]
J1S3R4[8--4-->12]
J2S0R2[0--3-->3]
J2S1R1[4..8--5-->9..13]
J2S2R3[9..13--3-->12..16]
J2S3R4[16--4-->20]
J3S0R1[2..6--2-->4..8]
J3S1R2[6..8--4-->10..12]
J3S2R4[12--4-->16]
J3S3R3[16..18--2-->18..20]
J4S0R3[0--3-->3]
J4S1R4[3--5-->8]
J4S2R1[9..13--4-->13..17]
J4S3R2[13..17--3-->16..20]



<그림 3-3> 4×4문제의 이웃해 Gantt Chart

3.4 tabu 파라메타

이웃해 생성단계에서 생성된 이웃해들의 개선을 통해 좋은해로의 이동을 위해 적용되는 tabu 파라메타에 대해서 설명한다. 설정된 tabu 파라메타는 알고리즘 단계에서 생성된 이웃해의 타부상태검사 알고리즘을 적용시 사용된다.

tabu list

tabu list는 이동의 속성을 저장하는 것으로, 탐색과정에서 tabu list의 크기 T 만큼 해의 순환을 제약할 수 있다. 본 연구에서는 T의 크기를 각 실행횟수마다 지역적으로 정하여 사용하는 동적 T를 사용한다. 하나의 실행횟수에서 생성된 초기해의 critical path를 이루는 critical 공정의 개수로 정한다. T의 갱신은 선입선출 방식을 이용한다.

tabu 속성

tabu 속성은 해의 이동을 제약하는 것으로 어떤 속성을 갖는 해의 이동을 막음으로써 해의 순환(cycling)을 방지하고 탐색속도를 빠르게 한다. 본 연구에서는 매 iteration마다 좋은 해를 생성하였을 때 삽입을 위해 선택된 공정과 삽입되는 위치에 있는 공정 및 이때 bounding된 각각의 시작시간과 종료시간을 고려하는 tabu 속성을 이용한다.

tabu state

본 연구에서는 tabu state검사를 위한 목적함수 값을 현재해의 총처리시간으로 하며, 어떤 움직임이 tabu 상태이지만 이를 해제하여 해의 움직임을 가능케 하는 기준으로서의 열망기준은 탐색한 가장 좋은 해(즉 최선해)의 총처리시간으로 한다.

3.5 제안된 TS와 CST에 의한 발견적 기법

탐색전략 및 파라메타에 대해 세부적으로 설명한 내용을 본 절에서는 종합화하여 제시한다. 즉, ILOG 프로그래밍 라이브러리를 이용하여 JSSP를 CSP로 모델링한 후 좋은해를 도출하고자, 초기해를 Minslack 알고리즘을 적용하여 생성하고, TS 적용단계에서 Tabu 파라메타의 조정을 통하여 해의 개선을 유도하기 위한 삽입이동과, one-step look- 본 연구가 제안하는 일정계획 알고리즘을 위해 필요로 하는 탐색전략 및 파라메back interchange 절차를 적용한 이웃해 생성 알고리즘을 제시하고, 생성된 하나의 이웃해에 대해서 타부상태 검사를 적용하는 단계로 최적화 알고리즘은 구성된다.

제안된 TS와 CST를 이용한 Job Shop 일정계획 알고리즘 단계는 다음과 같다.

<TS와 CST를 이용한 Job Shop 일정계획 알고리즘>

단계 1 : 초기화 단계
알고리즘을 진행하는 파라메터들로 Tabu 파라메타, 실행횟수, 종료조건을 초기화한다.

단계 2 : 초기해 생성 단계
초기해 생성 알고리즘에 의해 하나의 초기해를 생성한다. 생성된 초기해를 현재해와 최선해로 설정한다.

단계 3 : 이웃해 생성 단계(TS 기법 적용)
이웃해 생성 알고리즘을 통해 이웃해를 생성하고, 타부상태 검사 알고리즘을 따른다. 이 단계에서 탐색 효율의 증가를 위해 총처리시간의 도메인영역변화 기법을 적용하고, Backtrack 수를 정해준다.

단계 4 : 종료 조건을 만족하면 알고리즘은 종료되고, 그렇지 않으면 2단계부터 다시 반복한다.

4. Job Shop 벤치마킹 문제에의 적용

4.1 실험 방법

제안한 발견적 기법을 JSSP의 표준 벤치마킹 문제에 적용하여 얻어진 결과를 기존의 다른 연구들과 비교하여 제시한다. 실험에 필요한 파라메터로 제시되는 실행횟수(iteration 수)는 문제의 크기에 따라 정하여 실시하였다. 예를 들어, MT10(10×10)의 경우 작업이 10개이므로 실행회수는 10회가 된다. backtrack 수는 초기에 10회로 주어지고, 탐색 중에 1000회 backtrack 수를 초기 backtrack 수에 더하여 탐색을 강화하였다. tabu 파라메타는 3.4절에서 설명한 파라메타를 이용한다. 실험결과에서 backtrack 수는 실행횟수동안에 최적해를 발견하였을 때까지의 backtrack 수를 의미하며, 계산시간 또한 최적해를 발견하였을 때의 CPU시간(단위 초)을 나타낸다.

4.2 MT 벤치마킹 문제에의 적용결과

본 연구에서 제안한 TS와 CST에 의한 발견적 알고리즘을 Muth와 Thompson의 MT 벤치마킹 문제에 적용한 결과 <표 4-1>에서 보는바와 같이 비교적 짧은 해의 탐색시간으로 세 가지 문제에서 최적해를 얻었다. MT06, MT20문제의 경우에는 MT10과 비교할 때 비교적 적은 backtrack 수와 짧은 계산시간 내에 최적해를

<표 4-1> 제안된 발견적 기법의 MT문제 적용

벤치마킹 문제	n×m	최적해	제안된 발견적 기법		
			해	back track	cpu time
MT06	6×6	55	55	3	0.49
MT10	10×10	930	930	10632	1100.54
MT20	20×5	1165	1165	99	12.19

얻었다. MT10과 같이 문제가 커지는 경우에는 backtrack 수의 증가와 해의 탐색시간이 증가함을 알 수 있다.

<표 4-2>는 본 연구에서 제안된 발견적 기법과 ILOG[11]에서 JSSP를 해결하고자 제시한 CST를 적용한 기존의 방법인 Randomizing Algorithm 및 Dichotomizing binary search Algorithm과의 비교 결과를 보여준다. 이와 같이 비교하는 것은 동일한 컴퓨터 환경에서 구현하여 실험함으로써 본 연구에서 제안한 발견적 기법의 객관적인 수행도 평가를 제시할 수 있기 때문이다. 즉, 최적해를 얻기 위해 발생된 backtrack 수와 계산시간을 상호 비교함으로서 보다 명확하게 제안 알고리즘의 효율성을 제시 할 수 있다.

CST를 적용한 기존의 방법과의 비교결과, 본 연구에서 제안한 알고리즘과 Dichotomizing binary search Algorithm이 세 가지 MT문제에 대해 최적해를 구하였으나, Randomizing Algorithm은 MT10 문제에서 최적해와 다소 차이가 나는 결과를 보여주었다. 전체적으로 비교하였을 때, 본 알고리즘이 최적해를 탐색하는 계산시간 면에서 두 가지 제시된 알고리즘에 비해 다소 증가하였지만, backtrack 수에서 Randomizing Algorithm 보다는 작게 발생하였고, MT10문제의 경우 Dichotomizing binary search Algorithm과의 비교에서는 많은 차이를 보여주고 있다.

결론적으로 본 알고리즘은 수행도 면에서 Randomizing Algorithm 보다는 우수하고, Dichotomizing binary search Algorithm과는 계산시간 면에서는 다소 시간이 걸리지만, backtrack 수에서는 우수하다 할 수 있겠다.

<표 4-3>은 세 개의 MT문제에 대해 이전 연구의 다른 발견적 기법을 적용하여 얻어진 결과들과 본 연구에서 제안된 발견적 기법과의 비교표이다. JSSP를 해결하기 위한 이전 연구들 또한 다양한 발견적 기법을 제시하고 얻어진 최고의 결과들을 보여주고 있다. 계산시간은 이전 연구들에서 사용된 컴퓨터와 본 실험에서 사용된 컴퓨터 기종이 다르기 때문에 정확한 비교가 힘들어 제외하였고, 총처리시간만을 가지고 비교하였다. 각각의 비교 논문에서 적용한 주요 탐색 알고리즘은 표의 비고

<표4-2>제안된 발견적 기법과 CST를 적용한 기존의 방법과의 비교

벤치마킹 문제	최적해	Randomizing Algorithm			Dichotomizing binary search			제안된 발견적기법		
		해	BT	cpu time	해	BT	cpu time	해	BT	cpu time
MT 06	55	55	10	0.16	55	4	0.11	55	3	0.49
MT 10	930	996	1251	6.43	930	31011	155.28	930	10632	715.96
MT 20	1165	1165	1084	6.92	1165	40	1.98	1165	99	12.19

<표 4-3> MT문제에서 제안된 발견적 기법과 이전 연구와의 비교

비교 논문	MT06 (6×6)	MT10 (10×10)	MT20 (20×5)	비고
최적해	55	930	1165	
Dell'Amico & Trubian[7]	55	935	1165	TS
Balas et al[5]	55	940	1199	SB
Pezzella & Merelli[15]	55	930	1165	TSSB
Sabuncuoglu & Gurgun[16]	55	930	1165	Neural Networks
Sun, Batta & Lin[17]	-	930	-	TS
Wang & Zheng[19]	55	930	1165	GASA
Yamada & Nakano[21]	-	930	1178	CBSA
	-	930	1165	CBSA+SB
박병주[3]	55	930	1173	GA
이덕성[1]	55	966	1180	TS
제안된 발견적 기법	55	930	1165	CST+TS

Genetic Algorithm(GA), Simulated Annealing(SA), Tabu Search(TS) Tabu Search method and Shifting Bottleneck procedure(TSSB), Shifting Bottleneck procedure(SB), Branch and Bound(BB), Critical Block Simulated Annealing(CBSA)

란에 표기하였고, 표 하단에 약어의 원문을 적어 놓았다. 또한 표에서 "-"은 이전 연구논문에서 연구자가 주어진 MT 벤치마킹 문제에 대해서는 적용하지 않았음을 보여준다.

MT문제에서 10개의 작업들과 10개의 기계로 이루어진 MT10과 20개의 작업들과 5개의 기계로 이루어진 MT20과 같은 규모의 문제에서는 최적해를 얻었다. 그러나 MT10과 비교하여 문제를 이루고있는 작업과 기계의 개수가 증가하는 다소 큰 문제에서 제안된 발견적 기법의 효율성을 비교하고자 LA벤치마킹의 다양한 문제에 적용하여 실험하였다.

4.3 LA 벤치마킹 문제에의 적용결과

본 연구에서 제안된 발견적 기법을 Lawrence에 의해 제안된 LA문제 중 문제의 크기에따라 선정하여 적용하였다. <표 4-4>는 본 연구에서 제안한 발견적 기법에 의해 생성된 결과를 보여주고 있다. 비교결과 작업이 10개이고 기계가 5개인 10×5문제와 작업이 10개이고 기계가 10개인 10×10문제에 대해서는 최적해를 적절한 시간 내에서 구하였다. 그러나 문제가 커지는 경우 즉, 작업의 개수가 증가하거나 기계의 개수가 증가하는 문제에 대해서는 최적해를 얻지 못하였으나, 적절한 시간 내에서 근사해를 구할 수 있었다.

<표 4-5>는 본 연구에서 제안된 발견적 기법과 CST를 적용한 기존의 방법인 Randomizing Algorithm 및 Dichotomizing binary search Algorithm과의 비교 결과를 보여준다. 표에서 *는 사용되어진 컴퓨터에서 1시간 CPU time 내에도 해를 발견하지 못하는 것을 의미한다. 즉, Dichotomizing binary search 알고리즘은 LA24, LA27, LA40문제를 1시간의 계산시간 내에서 해결하지 못하였다. 그리고 Randomizing Algorithm은 모든 문제에 대해(LA32 제외) 적절한 시간 내에 근사해를 구하였지만 최적해와는 많은 차이를 보여주고 있다.

<표 4-4> 제안된 발견적 기법의 LA문제에의 적용

벤치 마킹 문제	n×m	최적해	제안된 발견적 기법		
			해	backtrack	cpu time
LA02	10×5	655	655	50	2.1
LA19	10×10	842	842	10103	840.5
LA24	15×10	935	949	10124	996.7
LA27	20×10	1235	1252	15191	3188.5
LA32	30×10	1850	1858	6559	2311.6
LA40	15×15	1222	1229	6978	1568.9

<표 4-5> 제안된 발견적 기법과 CST를 적용한 기존의 방법과의 비교

벤치 마킹 문제	최적해	Randomizing Algorithm			Dichotomizing binary search			제안된 발견적 기법		
		해	BT	cpu time	해	BT	cpu time	해	BT	cpu time
LA02 (10×5)	655	660	1142	11.31	655	53	0.77	655	50	2.1
LA19 (10×10)	842	851	1393	13.46	842	22732	81.4	842	10103	840.5
LA24 (15×10)	935	967	1283	24.66	*			949	10124	996.7
LA27 (20×10)	1235	1310	1202	23.61	*			1252	15191	3188.5
LA32 (30×10)	1850	1850	265	16.97	1850	13	22.8	1858	6559	2311.6
LA40 (15×15)	1222	1281	1223	20.54	*			1229	6978	1568.9

<표 4-5>에서 Randomizing Algorithm 과 Dichotomizing binary search Algorithm은 LA32문제에 대해서는 최적해를 구하였다. 이는 두 알고리즘이 이러한 유형의 문제에 대해서는 좋은해를 도출하는 알고리즘이라 할 수 있겠다. 그러나 제안된 본 알고리즘은 작업이 10개이고 기계가 10개인 문제까지는 최적해를 구하고, 문제를 이루는 작업과 기계의 개수가 커지는 문제에 대해서는 최적해에 근사한 해를 효율적인 시간(cpu time 1시간) 내에서 얻고 있음을 볼 수 있다. 따라서 문제가 커지는 경우 CST를 적용한 기존의 방법인 Randomizing Algorithm과 Dichotomizing binary search Algorithm 보다는 해의 탐색 면에서 효율적이라 할 수 있다.

<표 4-6>은 LA문제에 대해 본 연구에서 제안된 발견적 기법과 JSSP를 해결하기 위한 다른 기법의 이전 연구와의 비교결과를 보여주고 있다. 표에서 "-"은 이전 연구에서 연구자가 주어진 LA 벤치마킹 문제에 대해서는 적용하지 않았음을 보여준다.

결론적으로 본 연구에서 제안한 TS와 CST를 결합한 일정계획 알고리즘은 MT10, LA19문제인 10×10문제까지의 다소 작은 문제에 대해서는 빠른 시간 내에 최적해를 구하지만, 문제를 이루고있는 작업과 기계의 개수가 증가하는 다소 큰 문제에 대해서는 어느 정도의 효율적인 시간 내에 근사해를 구하므로 알고리즘 수행도 면에서 효율적이라고 할 수 있다.

<표 4-6> LA문제에서 제안된 발견적 기법과 이전 연구와의 비교

비교 논문	LA02 (10×5)	LA19 (10×10)	LA24 (15×10)	LA27 (20×10)	LA32 (30×10)	LA40 (15×15)	비고
최적해	655	842	935	1235	1850	1222	
Balas et al[5]	667	878	976	1272	-	1262	SB
Bierwirth[6]	-	-	-	1269	-	1252	GA
Mattfeld[12]	655	842	938	1236	-	1228	Genetic local search
Nowicki & Smutnicki[14]	655	842	939	1236	1850	1229	TS
Pezzella & Merelli[15]	655	842	938	1235	1850	1233	TS+SB
Sabuncuoglu & Bayiz[66]	704	882	992	1316	1850	1347	Beam search
Sabuncuoglu & Gurgun[16]	655	842	-	-	-	-	Neural Networks
Sun, Batta & Lin[17]	-	850	954	1277	-	1255	TS
Thomsen[18]	655	842	935	1235	-	1224	BB+TS
Werner & Winkler[20]	655	852	976	1318	-	1263	Reinsertion algo'
Yamada & Nakano[21]	-	-	935	1235	-	1228	CBSA + SB
제안된 발견적 기법	655	842	949	1252	1858	1229	CST + TS

GeneticAlgorithm(GA) ,Simulated Annealing(SA), TabuSearch(TS) Tabu Search method and Shifting Bottleneck procedure(TSSB) Shifting Bottleneck procedure(SB), Branch and Bound(BB) Critical Block Simulated Annealing(CBSA)

5. 결론

본 연구는 Job Shop 일정계획문제 분야에서 점점 대규모, 복잡화되는 현실에서 적절한 시간 내에 최적해에 근사한 해를 구하기 위한 발견적 기법을 제시하고자 하는 것이다. 이러한 문제를 해결하기 위해 최근 사용되는 AI기법의 CST를 기반으로 하는 ILOG를 이용하여 문제에 대한 모델링을 구현하였고, CST가 가지는 문제점인 전체 탐색공간에 대한 탐색으로 인한 시간적인 문제점을 해결하고, 해의 개선을 통해 좋은해로의 이동을 위해 발견적 기법으로 알려진 TS를 이용하였다.

TS를 적용함에 있어 효율적인 탐색을 위한 Tabu 파라메타를 설정하고, 초기해에서 이웃해를 생성하는 알고리즘 부분에서 해의 개선을 통해 좋은해를 얻고자 적용하였다. 또한 연구에서는 삽입이동 및 one-step look-back interchange 절차를 이용한 새로운 이웃해 생성방법을 제안하였고, 총처리시간 변수의 도메인 영역변화 기법과 backtrack 수를 조정하여 탐색의 효율성을 증가시키고, 탐색을 강화하였다.

연구에서 제안된 발견적 기법의 객관적인 수행도 평가를 위해 JSSP 벤치마킹 문제 중에서 3가지 MT문제인 MT06, MT10, MT20문제와, LA문제 중에서 문제의 크기에 따라 선정된 6가지 문제에 적용하여 실험하였다. 분석은 기존연구논문에서 연구자들이 제안한 발견적 기법과는 총처리시간 면에서 비교하였고, CST를 적용한 기존의 방법인 Randomizing Algorithm과 Dichotomizing binary search Algorithm과는 총처리시간, backtrack 수, 계산시간 면에서 비교하였다.

실험결과 연구에서 제안된 발견적 기법은 MT문제 등과 같이 비교적 적은 규모(작업과 기계의 개수가 10×10, 20×5문제)의 문제에 대해서는 최적해를 구하였다. MT문제에 적용된 이전 연구와의 비교결과에서는 몇몇 연구보다 우수한 해의 개선을 이루어 내었고, CST를 적용한 기존의 방법과의 비교에서는 Randomizing Algorithm보다 해의 탐색 면에서 우수하였으나, 탐색시간 면에서 Dichotomizing binary search Algorithm 보다는 다소 걸렸다. 연구에서 목적으로 하는 문제를 이루고 있는 작업과 기계의 개수가 증가하는 대규모 문제(MT10보다 작업 또는 기계의 개수가 많은)인 즉, LA문제 중 다소 규모가 큰 문제에 대해서는 CST를 적용한 기존의 방법보다는 효율적인 탐색시간 내에서 근사해를 구하였다. 또한 기존 연구와의 비교에서도 몇몇 연구보다 우수한 해를 도출하였다. 결론적으로 본 연구에서 제안된 발견적 기법은 수행도 면에서 JSSP의 대규모 문제를 해결하는데 효율적이며 우수하다 할 수 있겠다.

추후 연구과제로는 실제 생산현장에서 이루어지는 제약조건 등을 반영한 현실적인 상황에서의 대규모 문제에 제안된 발견적 기법을 적용하여 알고리즘의 효율성을 다시 검증하는 것이고, 본 연구에서 제안된 발견적 기법과 CST기법을 다른 메타 휴리스틱인 GA, SA와 결합한 발견적 기법과의 비교연구가 이루어져야 하겠다.

참고문헌

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

[2] 김기동, 우상복, 한형상, "자원제약을 고려한 조선산업에서의 탑재 일정계획에 관한 연구", *IE Interfaces*, vol.14, No.3, pp.218-226, 2001.

[3] 박병주, 김현수, "Job Shop 일정계획을 위한 유전알고리즘", *산업경영시스템학회지*, 제23권 제59집, pp. 11-20, 2000.

[4] Alexander, N., "Applying Local Search to Structural Constraints Satisfaction", *German National Research Center for Information Technology*, 1999.

[5] Balas, E., J.K. Lenstra and A. Vazacopoulos, "The One-machine Problem with Delayed Precedence Constraints and Its Use in Job Shop Scheduling", *Management Science*, Vol.41, No.1, pp.94-109, 1995.

[6] Bierwirth, C., "A generalized permutation approach to job-shop scheduling with genetic algorithm", *OR Spektrum*, 17(2-3), pp.87-92, 1995.

[7] Dell'Amico, M. and M. Trubian, "Applying tabu search to the job shop scheduling problem", *Annual of Operations Research*, vol.42, pp.231-252, 1993.

[8] Garey, M.R, John D.S, and Sethi, R, "The Complexity of flow shop and Job Shop Scheduling", *Math. Operations Research*, vol. 1, pp.117-129, 1976.

[9] Glover, F., "A user's guide to tabu search", *Annals of Operations Research*, vol 41, pp.3-28, 1993.

[10] ILOG, *ILOG Solver manual*, ILOG.

[11] ILOG, *ILOG Scheduler manual*, ILOG.

[12] Mattfeld, D.C., *Evolutionary Search and the Job Shop: Investigations on Genetic Algorithms for Production Scheduling*, Physica, Heidelberg, 1996.

[13] Michael, P., *SCHEDULING Theory, Algorithms and Systems*, Prentice Hall, 1995.

[14] Nowicki, E. and C. Smutnicki, "A Fast Taboo Search Algorithm for the Job Shop Problem",*Management Science*, Vol.42, No.6, pp.797-813, 1996.

[15] Pezzella, F. and E. Merelli, "A tabu search method guided by shifting bottleneck for the job shop scheduling problem", *European Journal of Operational Research*, 120, pp.297-310, 2000.

[16] Sabuncuoglu, I. and B. Gurgun, "A neural network model for scheduling problems", *European Journal of Operational Research*, 93, pp.288-299, 1996.

[17] Sun, D., R. Batta and L. Lin, "Effective Job Shop Scheduling through Active chain Manipulation", *Computers & Operations Research*, Vol.22, No.2, pp.159-172, 1995.

[18] Thomsen, S., "Metaheuristics combined with branch and bound(in Danish)", *Technical Report*, Copenhagen Business School, Copenhagen, Denmark, 1997.

[19] Wang, L. and D.Z. Zheng, "An effective hybrid optimization strategy for job-shop scheduling problems", *Computers & Operations Research*, 28, pp.585-596, 2001.

[20] Werner, F. and A. Winkler, "Insertion techniques for the heuristic solution of the job-shop problem", *Discrete Application Math'*, 58(2), pp.191-211, 1995.

[21] Yamada, T. and R. Nakano, "Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search", *Kluwer academic publishers*, MA, USA, pp.237-248, 1996.