「정보시스템연구」 제25권 제1호
한국정보시스템학회
2016년 3월, pp. 159~182
http://dx.doi.org/10.5859/KAIS.2016.25.1.159

후보순위 기반 타부 서치를 이용한 제약 조건을 갖는 작업 순서결정 문제 풀이*

정 성 욱** · 김 준 우***

〈목 차〉

Ⅰ. 서론

단일기계 순서결정 문제(single machine sequencing problem, SMSP)는 조합최적화(combinatorial optimization)의 일종으로서, 가용할 수 있는 기계 1대에 투입되는 작업들의 작업 순서를 결정하는 문제이다. 각 작업은 처리시간(processing time)과 납기일(due date)을 가지고 있으며 우선순위와 같은 다른 특징들을 가질 수도 있다. 작업들의 나열 순서 중, 처리시간(makespan)이나 납기지연과 같은 목적함수의 값을 최소화하는 것을 최적해(optimal solution)라고 하며, 이들은 제조현장의 업무 효율성 향상을 위해 활용될 수 있다(Baker, 1974; Sule, 1996).

해 공간이 이산적인 조합최적화 문제의 특성상, 완전열거법(exact method)이나 수리적 최적화(mathematical programming) 등을 사용하여 최적해를 찾을 수 있는 경우도 있으나, 문제의 규모가 커질 경우 정확한 최적해를 찾는 시간이 기하급수적으로 증가한다는 문제가 존재한다. 예를 들어, 생산업체에서 납기일이 하루가 남아 있을 때, 이틀에 걸쳐 정확한 최적해를 찾을 수 있는 방법은 무의미할 수 밖에 없다. 때문에 비교적 신속하게 최적해 또는 근사 최적해(near optimal solution)를 찾을 수 있는 휴리스

* 이 논문은 동아대학교 교내연구비 지원에 의하여 연구되었음.
** 동아대학교 일반대학원 산업경영공학과, infinite03@hanmail.net
*** 동아대학교 산업경영공학과 조교수(교신저자), kjunwoo@dau.ac.kr

틱 또는 메타 휴리스틱 해법들에 대한 연구와 활용이 활발하였다(Johnsom and Montgomery, 1974; Ahmed and Fisher, 1992; Hopper and Turton, 2001; Lee, 2005).

휴리스틱의 예로는 최단시간(shortest processing time, STP)규칙, 최단납기(earliest due date, EDD)규칙, 시간대비비용(cost over time, COVERT)규칙 등과 같은 작업할당규칙(dispatching rule)들이 잘 알려져 있으나, 일반적으로 이러한 휴리스틱 해법들은 특정 문제에만 적용이 가능하고, 개발하는 과정이 까다롭기 때문에 새로운 문제를 신속하게 해결하는 것이 어려울 수 있다(Blackston et al., 1982; Sule, 1996). 한편, 메타 휴리스틱 해법으로는 유전알고리즘(genetic algorithm)(Holland, 1975), 타부서치(tabu search)(Glover, 1990), 시뮬레이티드 어닐링(simulated annealing)(Kirkpatrick et al., 1983), 입자군집(particle swarm)(Kennedy and Eberhart, 1995) 등이 있으며, 이들은 조합최적화 문제 풀이를 위한 일반적인 전략을 제시하여 이를 다양한 문제에 적용할 수 있다는 장점을 갖는다. 그러나 실제 적용 시에는 주어진 문제에 적합한 해 변형 방법이 필요하여, 이들 메타 휴리스틱 해법 역시 특정 문제에 종속적이어서 그 범용성이 기대보다 높지 않음이 지적되고 있다(Chakhlevitch and Cowling, 2008).

일반적으로 조합최적화 문제는 조합(combinatorial) 문제, 순서결정(sequencing) 문제 및 그룹화(grouping) 문제로 분류되며, 이 중에서도 순서결정 문제는 다른 것들에 비해 해를 표현하거나 변형하는 과정이 비교적 까다롭다. 예를 들어, 대표적인 조합 문제인 배낭 문제의 경우 각 물품의 포함여부를 0과 1로 표현한 이진 문자열로 해를 표현할 수 있지만, 순서결정 문제의 경우 이러한 표현이 곤란하다. 나아가, 물품들의 포함 여부가 서로 독립적인 배낭 문제에서 특정 물품을 포함 또는 제외시킬 때 다른 물품들의 영향을 비교적 적게 받는 것과 달리, 순서결정 문제에서는 특정 순서에 특정 항목을 배치할 경우, 다른 항목이 동일한 순서를 점유할 수 없고, 각 항목이 정확히 한 개의 순서만을 점유해야 하는 점 등이 고려되어야 한다. 나아가, 부가적인 제약 조건이 추가되는 경우 순서결정 문제의 복잡도(complexity)는 극히 높아지는 것으로 알려져 있다(Lawler, 1978; Lenstra and Kan, 1978; Kowalczyk, 1997).

이에, 본 논문에서는 전통적인 단일기계 작업 순서결정 문제와 같은 순서결정문제에 제약조건이 추가된 상황에서도 활용할 수 있는 타부 서치 방법을 개발하여, 첫째, 높은 복잡도를 갖는 조합최적화 문제를 위한 해법을 제시하고, 둘째, 기존 메타 휴리스틱 해법들의 범용성을 높이는데 기여하고자 한다. 구체적으로, 새로운 타부 서치 방법의 개발을 위해 본 논문에서는 최근 유전 알고리즘에서 활용되었던 후보순위 방식(candidate order scheme, CO-scheme)(Kim, 2015; Kim, 2016)을 타부 서치에 적합한 형태로 변형 및 적용하여, 높은 범용성을 갖는 후보순위 기반 타부 서치(candidate order based tabu search, COTS) 알고리즘을 개발하였다. 결과적으로 COTS를 활용 시, 현업에서 다양한 순서결정 문제를 보다 효과적으로 다룰 수 있을 것이다. 나아가, 본 논문에서는 COTS를 단일기계 작업 순서결정 문제에 적용하여 그 성능을 평가하였으나, 일반적으로 순서결정 문제

의 응용 분야가 전통적인 제조 현장의 일정계획(Sule, 1997) 뿐만 아니라, 교통수단의 경로관리(Laporte, 1992) 및 전자상거래에서의 의사결정(Kovalyov et al., 2010; López-Locés et al., 2015) 등에 이르기까지 매우 다양함을 생각해 볼 때, 본 연구가 상당한 파급 효과를 가질 것으로 기대된다.

본 논문의 구성은 다음과 같다. 2장에서는 먼저 타부 서치 알고리즘과 관련된 기존 문헌들에 대해 간단히 살펴본 후, 새로운 타부 서치 알고리즘을 설계하고 개발하는데 사용한 후보순위 방식(candidate order scheme)에 대해 설명한다. 3장에서는 본 논문에서 제안하는 COTS 알고리즘의 구조와 절차에 대해 설명하고, JAVA 언어를 이용한 실제 구현 내역을 소개한다. 이어서 4장에서는 COTS의 성능을 평가하기 위해 이를 단일기계 작업순서 결정문제에 적용하여 얻은 실험결과에 대해 토의하고자 하며, 마지막으로 5장에서는 결론 및 추후 연구과제를 제시한다.

Ⅱ. 연구 배경

2.1 타부 서치 알고리즘

일정계획(scheduling)이란 수행해야할 여러 작업들의 시작 및 종료 시각을 결정하는 것으로, 특정한 조건하에서 설비・기계・인원 등을 최대한 효과적으로 사용하는 것을 목표로 한다. 나아가, 전통적으로 일정계획은 주로 제조업, 건설업과 같은 2차 산업에서 많이 다루어졌으나, 최근에는 상업, 금융업 등 서비스업에서도 많이 적용되고 있는 상황이다.

2차 세계대전 이후 제조업 현장에서는 대량생산시스템의 체계가 등장하였다. 이와 함께 일정계획의 중요성이 높아지면서 작업할당규칙 등과 같은 휴리스틱에 의한 일정계획이 많이 연구되었다. 아울러, 1970년대부터는 JIT (just-in-time) 시스템 개념이 제조업 분야에서 확산되면서 이를 일정계획에 적용하기 위해 조기 및 지연완료를 함께 고려한 연구가 늘어났다 (Koh, 2014). 한편, 기술과 생산 시스템의 발전으로 인해 일정계획 문제가 복잡해지면서 작업할당규칙으로 해결하기 곤란한 상황들이 늘어났으며, 특히, 일정계획 문제의 상당수가 NP문제에 해당하기 때문에 1970년대 이후부터는 일정계획 문제에 메타 휴리스틱을 적용하는 방법에 관한 연구가 활발하였다.

메타 휴리스틱이란 휴리스틱 기법 보다 상위 수준의 해법으로, 흔히 자연현상을 모방하여 규칙(rule)과 임의성(randomness)을 결합한 형태의 알고리즘을 지칭한다(김여근 외, 1997). 그 중에서도 Glover(1990)가 제안한 타부 서치는 고전적인 국소 탐색(Lenstra, 1997) 방법에 몇 가지 탐색 전략을 추가하여 탐색 능력을 크게 개선한 것으로, 한 번에 해 한 개씩을 탐색해 나가는 대표적인 점 기반 탐색(point based search) 기법이다. 타부 서치는 좋은 해를 탐색하기 위해 현재 해와 인접한 이웃 해(neighbor)들의 목록을 생성한 후, 이들 중 가장 좋은 해로 이동하는 것을 반복하며, 이 때 단기 메모리에 해당하는 타부 리스트에 해당하는 이웃 해로의 이동을 지양함으로써 탐색 과정에서 국소 최적해(local optima)에로의 수렴을 방지한다. 나아가, 상당 기간의 탐색에도 불구하고 해의 개선

이 이루어지지 않았을 경우, 다양화(diversification) 작업을 통해 해 공간에서 탐색이 이루어지지 않은 곳으로 이동하여 추가적인 해의 개선을 모색하기도 한다.

타부 서치를 이용한 조합최적화 문제 풀이에 대해서는 과거 다양한 연구들이 이루어졌으며, 그 응용 분야는 생산 현장의 일정계획 문제(Colin, 1993; Nowicki and Smutnicki, 1996; Pezzella and Merelli, 2000; Meeran and Morshed, 2012), 차량경로 문제(Gendreau et al., 1994; Toth and Vigo, 2003; Cordeau and Maischberger, 2012), 시간표(time tabling) 문제(Hertz, 1991; Lü and Hao, 2010) 및 외판원 문제(traveling salesman problem)(Gendreau et al., 1998; Misevičius, 2015) 등으로 매우 다양하다. 타부 서치는 이러한 문제들을 해결하는데 있어 우수한 성능을 보였으며, 효과적으로 국소최적해를 회피하고, 비교적 신속하게 좋은 해를 탐색할 수 있다는 장점이 있다.

그러나, 타부 서치 역시 다른 메타 휴리스틱 방법들과 마찬가지로 특정 문제를 해결하기 위해서는 그에 맞는 해 표현 방법, 이웃해 구조 및 탐색 방법의 개발이 선행되어야 하며, 이러한 점으로 인해 현장에서는 종종 다양한 문제에 타부 서치를 활용하는데 어려움을 겪기도 한다. 특히, 조합최적화 문제에는 상황에 따라 제약조건이 추가되는 경우가 있는데, 예를 들어 작업 순서결정 문제에 특정 작업들 간의 선행관계(precedence)에 대한 제약이 가해지는 경우, 나머지 상황은 기존과 동일하더라도 새로운 해법을 개발해야 하는 불편함이 따를 수 있다. 이에, 본 논문에서는 이러한 추가적인 제약의 유무에 무관하게 작업 순서결정 문제를 범용적으로 다룰 수 있는 타부 서치 해법을 개발하여, 보다 일반적인 목적으로 활용할 수 있는 메타 휴리스틱 방법의 개발에 기여하고자 한다.

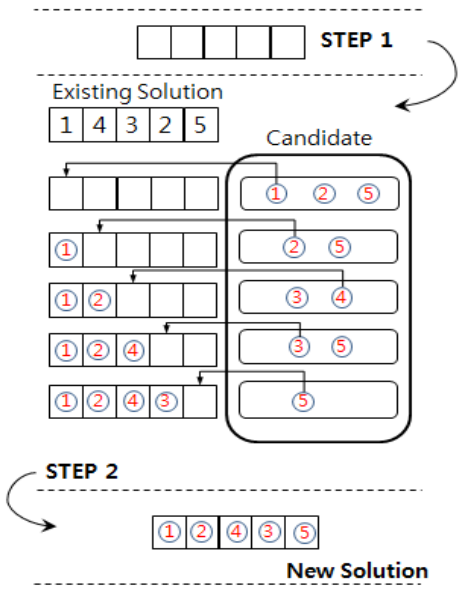
2.2 후보순위 방식

앞에서 타부 서치의 주요 절차 중 하나가 현재 해의 이웃해들을 식별하고 이들 중 가장 좋은 것을 선택하는 것임을 설명하였다. 본 논문에서 다루는 것과 같은 순서결정 문제의 경우 일반적으로 사용되는 이웃해 생성 방법은 크게 두 가지로, 먼저 현재 해를 의미하는 시퀀스(sequence) 내에서 특정 작업(들)의 위치를 현재와 다른 곳으로 이동시키는 삽입(insertion) 방식(Gendreau et al., 1998; Cordreau et al., 2008; Gao et al., 2013)과 특정 작업(들)의 위치를 서로 뒤바꾸는 교환(swapping) 방식(Cordreau et al., 2008; Gao et al., 2013; Misevičius, 2015)을 들 수 있다. 이들은 과거 다양한 문헌에서 이용되어졌으나, 선행 제약이 있는 경우, 삽입 또는 교환 과정에서 제약을 위배하는 실행불가능해(infeasible solution)가 발생할 수 있어, 이를 방지하기 위해 복잡한 형태의 이웃해 생성 절차가 필요해진다.

이러한 문제를 해결하기 위해 본 논문에서는 유전 알고리즘에서 이러한 문제를 해결하기 위해 사용되었던 후보순위방식(candidate order scheme)을 타부 서치에 적용하고자 한다. 후보순위방식이란 순서결정 문제의 해 한 개를 생성할 때, 첫 번째 위치부터 항목을 한 개씩 순차적으로 추가하되, 각 위치에서 추가될 수 있는 항목들을 후보로 식별하고 이들 중 적절한 것을 선택하는 방법이다. 후보순위방식(candidate

order scheme)이란 기존해(existing solution)의 우선(early) 순위를 인정하여 이웃해를 생성하는 방법이다. Kim(2015)은 job shop 일정계획 문제에 후보순위 방식을 적용한 유전알고리즘(candidate order based genetic algorithm, COGA)을 적용하여 우수한 해를 찾을 수 있다는 것을 보였다. 특히, COGA에서는 부모 해에서 위치가 앞쪽인 후보를 선택할 경우, 부모 해의 특성이 상속되어 이를 교차(crossover) 연산으로 사용할 수 있고, 부모 해에서 위치가 뒤쪽인 후보를 선택할 경우, 부모 해와 상이한 특성이 발생하여 이를 통해 돌연변이(mutation) 연산으로 사용할 수 있음을 제시하였다.

세부적으로, COGA에서 부모 해에 대한 자손 해는 <그림 1>과 같이 생성된다. 먼저, STEP 1에서 비어있는 염색체 하나를 생성하고, 다음으로 STEP 2에서는 할당이 가능한 후보 항목들을 모아둔 후보목록(candidate list)에서 부모 해의 우선순위를 참조하여 비어있는 염색체에 항목을 하나씩 할당한다. 이 때, 교차를 수행하고자 할 경우에는 부모 해에서 가장 위치가 앞에 있는 후보를, 돌연변이를 수행할 경우에는 가장 위치가 뒤쪽인 후보를 선택한다. 모든 유전자가 할당이 될 때까지 STEP 2를 반복하면 새로운 해 하나가 생성된다. 이러한 후보순위방식의 유전 연산자는 순서교차(order crossover, OX)나 부분사상교차(partially mapped crossover, PMX) 등과 같은 종래의 연산자들에 비해 그 의미가 매우 직관적이며 구현이 쉽다는 등의 장점을 갖는다.



<그림 1> 후보순위 방식을 이용한 해 생성

COGA의 경우, 종래의 유전 알고리즘들과 달리 추가적인 제약 조건이 있는 상황에서도 다양한 순서결정 문제들을 유연하게 다룰 수 있는데, 그 이유는 특정 위치에 대한 할당 시 후보목록을 생성할 때 이러한 제약들을 고려하기 때문이다. 다시 말해, 이미 할당된 항목 또는 할당된 적은 없으나 제약 조건으로 인해 현재 위치에 할당이 불가능한 항목들의 경우, 후보목록에서 사전에 제외함으로써, 후보목록에 있는 항목들 중 어떤 것들을 선택하여 할당하더라도 실행가능해(feasible solution)가 생성됨이 보장되고, 후보 항목 중 적절한 것을 선택하여 새로운 해에 기존 해와 비슷하거나 상이한 특성을 부여할 수 있다는 것이 후보순위 방식의 핵심이다.

Ⅲ. 후보순위 기반 타부 서치

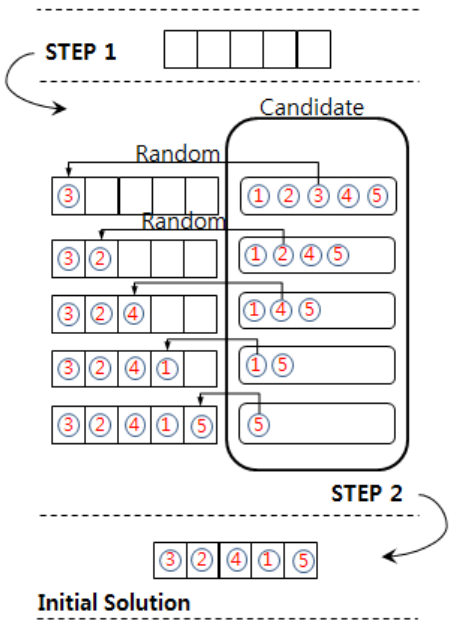
3.1 알고리즘 개요

1) 해의 표현 및 초기해 생성

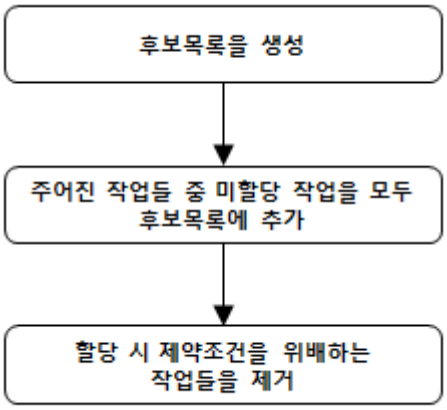
메타 휴리스틱 방법을 적용하기 위해서는 먼저, 해를 간단한 형태로 표현하는 것이 필요하며, COGA에서는 후보 항목들이 부모 해에서 몇 번째 위치를 점유하는지를 편리하게 조사하기 위해 위치 나열 표현(position listing representation)이 사용되었다(Kim, 2015; Kim, 2016). 위치 나열 표현이라는 각 항목 1, 2, ..., n에 대하여 항목의 위치를 나열하여 항목들의 시퀀스를 표현하는 것으로, 본 논문에서도 COTS의 해 표현 방식으로 위치 나열 표현을 선택하였다.

COTS에서 위치 나열 표현 형태의 해를 생성할 때는 앞서 설명한 후보순위 방식이 이용되며, 그 중에서도 초기해는 <그림 2>와 같은 절차를 통해 생성된다. 먼저, STEP 1에서 비어있는 해를 생성하고, 다음으로 STEP 2는 제약 조건에 위배되지 않고 할당 가능한 후보들을 후보목록에 포함시킨 후, 아직까지 비교 대상이 되는 기존 해가 없기 때문에 임의의 후보 한 개를 선택하여 할당하는 것을 반복한다. 이러한 작업을 주어진 항목의 개수만큼 반복할 경우 임의의 초기해가 생성된다.

후보목록은 <그림 3>과 같은 과정을 통해 생성되는데, 이를 보면 처음에 문제에서 주어진 항목들 중 앞에서 할당이 되지 않은 작업들을 후보목록에 추가한 후, 이들 중 현재 위치에 할당 시 제약조건을 한 개 이상 위배하는 것들을 제외한다.



<그림 2> 초기해 생성 과정



<그림 3> 후보목록 생성 과정

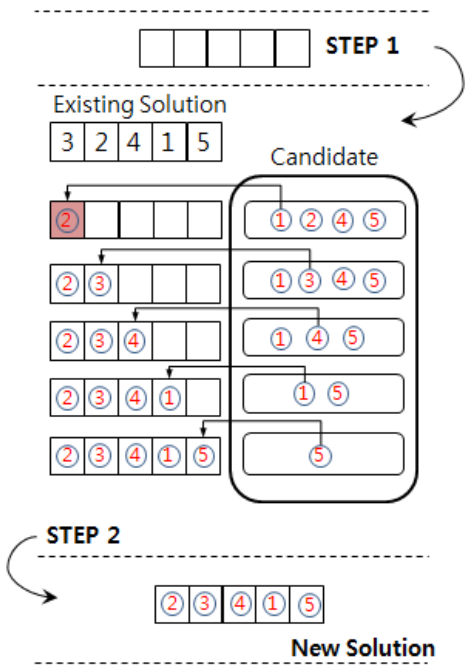
2) 이웃해의 생성 및 이동

타부 서치에서 이웃해란, 현재 해와 대부분의 특성이 유사하나 일부 내지 한 가지 특성에

서 차이를 보이는 해를 의미한다. 이러한 이웃해는 해 공간에서 현재 해와 인접한 위치를 점하고 있으며, 타부 서치를 비롯한 국소 탐색 기법들은 이웃해 중에서 좋은 것을 선택하여 이동하여 해를 개선하는 과정을 반복하게 된다.

종래 타부 서치에서 삽입이나 교환을 통해 이웃해를 생성한 것과 달리, 본 논문에서는 후보순위 방식을 이용하여, 해를 생성하는 과정에서 후보들 중 기존 해에서 가장 앞쪽 위치를 점하는 항목을 선택하여 할당하되, 한 곳에서만 다른 위치를 점하는 항목이 선택된 것을 이웃해로 정의한다. 즉, i번째 이웃해는 i번째 항목을 선택할 때 후보목록의 항목들 중에서 임의로 1개를 할당하고, 나머지 항목들은 후보 중 현재 해에서 가장 빠른 위치에 등장하는 것을 선택하여 생성된다. 예를 들어, <그림 4>는 현재 해가 [3 2 4 1 5]일 때, 첫 번째 이웃해를 생성하는 과정을 보여주며, 첫 번째 칸에 대한 후보 항목들인 1, 2, 3, 4, 5중, 현재 해에서는 항목 3이 가장 빠른 위치를 점유하고 있으나, 해의 변형을 위해 이를 배제하고 항목 2가 임의로 선택된 상황을 나타내고 있다. 두 번째 위치에 대한 후보는 이미 할당된 2를 제외한 1, 3, 4, 5이며, 이들 중 기존 해에서 가장 빠른 위치에 등장하는 3이 선택되었다. 이러한 과정을 반복할 경우, 새로운 해의 3~5번째 위치에는 각각 항목 4, 1, 5가 선택되어 결과적으로 [2 3 4 1 5]를 현재 해의 첫 번째 이웃해로 생성한다.

이웃해들의 목록이 생성되면 이들의 목적함수 값을 평가하여 그 중 가장 좋은 것을 선택해야 한다. 본 연구에서는 단일기계 작업 순서결정 문제를 다루기 위하여, 총 체류시간(total flow time, TFT)을 목적함수로 사용하였다.



<그림 4> 이웃해 생성 과정

TFT는 각 작업들이 작업장에서 머무른 시간 F_i들의 총합을 의미하며, 작업 i의 작업시간을 p_i, 작업 시퀀스에서 작업 i의 직전 선행 작업의 종료시간을 PC_i라 할 때, F_i는 (1)과 같이 산출된다. 단, 작업 i가 시퀀스 내에서 최초 작업일 경우 PC_i=0이 된다.

$$F_i = PC_i + p_i \tag{1}$$

작업이 총 n개 존재하는 경우, TFT는 (2)와 같이 산출되고, 본 논문에서는 TFT를 최소화하는 작업 시퀀스를 찾는 것이 목적이 된다.

$$TFT = \sum_{i=1}^{n} F_i \tag{2}$$

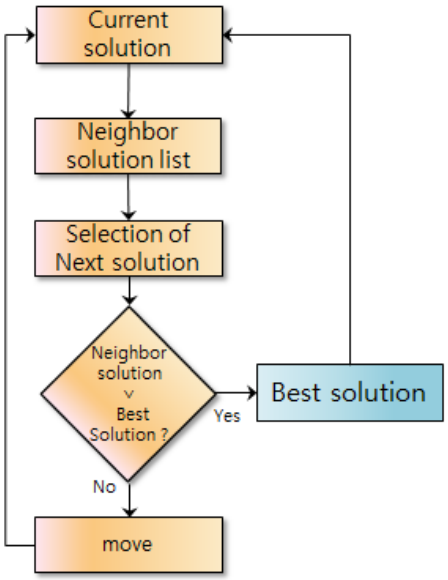
따라서, 현재 해에 대한 이웃해들이 생성되면, 이들 각각의 TFT를 계산하여 그 중 가장 작은 값을 갖는 것을 선택하여 이동하되, 만약 현재 해와 이웃해의 목적함수가 같은 경우에는 이동하지 않도록 하였다.

3) COTS 탐색 과정

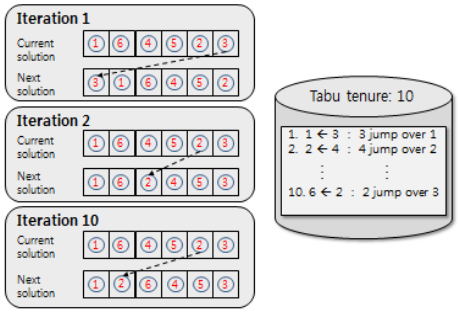
<그림 5>는 COTS에서 해가 이동해 가는 과정을 도식화하여 보여준다. 먼저, 현재 해에 대한 이웃해를 생성하여 이웃해 목록에 저장한다. 이웃해 목록에는 작업의 수가 n개인 경우 $n-1$ 개의 이웃해가 저장되며, 그 이유는 마지막 n번째 위치에 대한 후보 항목이 1개 뿐이기 때문에 n번째 후보는 현재 해와 동일하기 때문이다.

생성된 이웃해들에 대해서는 목적함수 TFT를 산출한 후, 이 값이 가장 작으면서 타부에 위배되지 않는 것을 선택하여 이동하되, 가까운 미래에 유사한 해를 다시 탐색하지 않도록 선택된 이웃해의 특성을 타부목록에 저장한다.

타부목록에는 특정 이동 시 기존 해와 비교하여 어떤 항목들의 위치에 서로 역전이 일어났는지를 기록하며, 예를 들어 <그림 6>의 Case 1에서는 이웃해를 생성하면서 작업 3이 작업 1을 역전하는 현상이 발생하였다. COTS에서는 이를 (1, 3)과 같은 작업 2개의 순서쌍으로 표시하고, 당분간 작업 1이 작업 3을 다시 앞지름으로써 이러한 해의 변형을 되돌리는 것을 금지한다.



<그림 5> 해의 이동

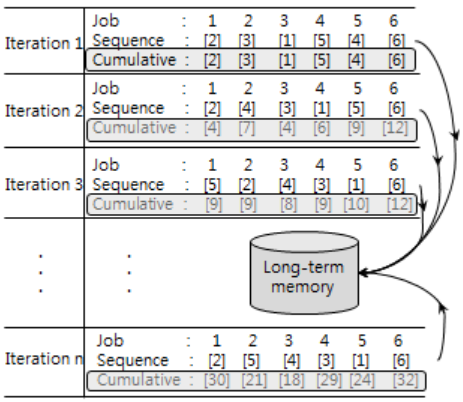


<그림 6> 타부목록 예시

만약 선택된 이웃해가 기존의 통산 최적해(best solution)보다 더 좋은 목적함수 값을 갖는다면 통산 최적해로 저장된다. 통산 최적해의 개선이 이루어 질 때 마다 비개선 이동 횟수를 0으로 초기화를 하며, 반대로 개선이 이루어지지 않을 때 마다 비개선 이동 횟수를 1씩 증가시킨다. 이 때 비개선 이동 횟수가 사전에 정의

된 기준값보다 크다면 다양화 전략이 실행된다.

COTS는 다양화 전략의 실행을 위해 각 작업들에 대하여 최초 탐색부터 이때까지 이동해 온 해들에서 자신이 점유했던 순서의 총합을 지속적으로 갱신하여, 이를 장기 메모리로 활용한다. <그림 7>은 이러한 장기 메모리의 갱신 과정의 예시이며, 실제 다양화를 실시할 때는 위치 총합이 가장 큰 것을 우선적으로 시퀀스의 앞쪽에 배치하여 이때까지 탐색해왔던 해들과는 상이한 양상의 해를 생성한 후, 여기서부터 탐색을 진행하도록 하였다.



<그림 7> 장기 메모리의 축적

3.2 세부 구현 내역

<표 1>은 Java 언어로 구현한 COTS의 전체 구조를 보여준다. COTS의 실행을 위해서는 종료 조건으로 사용하기 위한 최대 반복 횟수(max_iteration), 타부목록의 크기(tabu_tenure), 최대 비개선 횟수(max_cold_period) 3개의 매개 변수가 필요하며, 알고리즘에서 n은 문제에서 주어진 작업의 개수이다. 나아가, 3~10번 줄에서는 필요한 변수들이 선언되어 있으며, items, constraint는 각각 작업과 제약조건들의 집합이다. 또한, cur_sol, next_sol, best_sol은 각각 현재 해, 이웃 해 중 선택된 다음 해, 통산 최적해를 저장하기 위한 변수들이며, 단기 메모리에 해당하는 tabu_list에는 작업 2개의 순서쌍 형태로 표현되는 타부들의 목록을, 장기 메모리에 해당하는 lt_memory에는 매 이동 시마다 각 작업들의 점유 위치 총합을 저장한다. cold_period는 탐색 도중 연속해서 통산 최적해의 개선이 없었던 기간을 저장하는데 사용된다.

11~13번 줄은 초기화 절차에 해당하며, initialize(n)는 n개의 작업들을 이용하여 임의의 해를 생성하는 함수이다. 초기해는 통산 최적해로도 등록되며, 이후 현재 해의 내용을 장기 메모리에 반영하고, 14번 줄에서부터는 정해진 반복 횟수만큼 탐색을 반복한다.

매 반복 시에는 bestNeighbor(cur_sol) 함수를 이용하여 현재 해의 이웃해 중 가장 좋은 목적함수 값을 갖는 것을 선별하고, 이를 next_sol에 저장한다. 만약 next_sol의 목적함수가 통산 최적해보다 우위인 경우, 통산 최적해 best_sol를 갱신하면서 비개선 횟수를 0으로 초기화한다. 반면, next_sol이 통산 최적해보다 좋은 목적함수를 갖지 않는다면 비개선 횟수만 1씩 증가하게 된다. 23~26번 줄은 실제 해의 이동이 발생하는 부분이며, 비개선 횟수가 기준값 max_cold_period 보다 작은 경우, next_sol 과 관련된 타부를 타부목록에 추가하고, next_sol을 현재 해로 삼는다. 비개선 횟수가 충분히 크다면 next_sol과 무관하게 diversify (lt_memory) 함수를 실행시켜 다양화를 실시, 해공간의 새로운 영역에서 탐색을 계속하도록 한다.

<표 1> COTS 전체 실행 절차

```
1.COTS_search(int max_iteration, int tabu_tenure, int max_cold_period) {
2.          n = the number of items
3.          items[] = set of items
4.          constraint[] = set of constraints
5.          cur_sol = null
6.          next_sol = null
7.          best_sol = null
8.          tabu_list = null
9.        lt_memory[i] = 0        i = 1, 2, ..., n
10.         cold_period = 0
11.         cur_sol = Initialize(n)
12.         best_sol = cur_sol
13.         update_lt_memory(cur_sol)
14.         for(i=1; i<= max_iteration; i++) {
15.                     next_sol = bestNeighbor(cur_sol)
16.                     if (next_sol.TFT <= best_sol.TFT) {
17.                                 best_sol = next_sol
18.                                 cold_period = 0
19.                     }
20.                     else {
21.                                 cold_period++
22.                     }
23.                     if (cold_period < max_cold_period) {
24.                                 update_tabu_list(cur_sol, next_sol)
25.                                 cur_sol = next_sol
26.                                 update_lt_memory(cur_sol)
27.                     } else {
28.                                 cur_sol = diversify(lt_memory)
29.                                 tabu_list = null
30.                                 cold_period = 0
31.                     }
32.         }
33. }
```

<표 1>에서 initialize(n)으로 표현된 초기해 생성 과정에 대한 의사코드는 <표 2>에서 확인할 수 있다. 기본적인 구조는 최초에 모든 작업들의 할당 여부를 미할당(false)으로 설정하고, 텅 빈 해 1개를 생성한 다음 작업 개수 n회 만큼 할당을 반복하는 것이다. 할당 작업을 위해서는 먼저 후보 목록 candidate_list에 현재 미할당 상태인 작업들을 저장한 후, 이들 중 현재 위치에 할당 시, 위배되는 제약 조건이 있는 것들을 제외시킨다. candidate_list에 남아 있는 후보들 중에서는 어떤 것을 선택하더라도 실행가능해가 만들어지며, 초기해 생성 시에는 이 후보들 중 임의의 1개를 선택하여 할당한 후, 할당된 작업의 할당여부를 할당(true) 상태로 바꾼다. 이러한 할당 작업을 n회 반복 시, 임의의 해 1개가 생성되며, 이 해는 initialize(n) 함수에 의해 반환된다.

<표 2> 초기해 생성(Initialize)

```
1.Solution Initialize(int n) {
2.          item[i].isAssign = false, i = 1, 2, ..., n
3.          sol[i] = null, i = 1, 2, ..., n
4.          for(i=1; i<=n; i++) {
5.              candidate_list = set of unassigned items
6.             violate_list = set of unassigned items whose predecessors are not assigned
7.              candidate_list = candidate_list - violate_list
8.              next_item = a random element of candidate_list
9.              sol[i] = next_item
10              item[next_item].isAssign = true
11.         }
12.         return sol;
13. }
```

<표 3>은 <표 1>에서 update_lt_memory()로 표현된 장기 메모리 갱신 과정을 의미하며, 앞에서 설명한 바와 같이, 해 1개를 매개 변수로 전달하면, 각 작업에 대한 위치 총합이 누적되어 간다.

<표 4>는 <표 1>의 bestNeighbor()에 해당하는 이웃해 생성 및 선택 과정에 대한 의사코드이며, 매개 변수로 전달된 해 sol에 대하여 총 n-1개의 이웃해를 생성하고, 이들 중 목적함수가 가장 우수한 것을 선택하여 반환하도록 되어 있다. 특히, upSet(sol, j) 함수는 해 sol을 참조하여 새로운 해 하나를 생성하되, 다른 위치에서는 모두 후보 항목 중 sol에서 가장 빠른 것을 선택하여 sol과 유사한 특성을 부여하나, j번째 위치에서만 sol에서 가장 빠른 것이 아닌 다른 작업을 선택하여 sol의 이웃해를 반환하는 역할을 수행한다. 이러한 upSet(sol, j) 함수는 후보순위 방식에 기반한 이웃해를 정의하고 활용하는데 핵심적인 역할을 담당하며, 세부적인 내용은 아래에서 추가로 살펴보기로 한다.

<표 3> 장기 메모리 갱신(update_lt_memory)

```
1. void update_lt_memory(Solution sol) {
2.          for(i = 1; i <= n; i++)          {
3.                      lt_memory[i] = lt_memory[i] + order of item i in sol;
4.          }
5. }
```

<표 4> 최고 이웃해 선택(bestNeighbor)

```
1. Solution bestNeighbor(Solution sol) {
2.          for(j=1; j<=n-1; j++) {
3.                      Neighbor[j] = upSet(sol, j);
4.          }
5.          best_neighbor = the element of Neighbor with minimum TFT;
6.          return best_neighbor;
7. }
```

<표 5>는 <표 4>에서 upSet() 함수로 표현된 개별 이웃해 생성 과정의 세부 절차들을 보여주고 있다. 앞에서 설명한 바와 같이 이 함수의 역할은 주어진 기존 해 sol을 이용하여 후보순위 방식으로 이웃해 한 개를 생성하되, k번째 위치에서만 기존 해와 다른 방식으로 할당할 작업을 선택하여 기존 해와 상이한 특성을 부여하고, 나머지 위치들에서는 기존 해와 같은 방식으로 작업을 할당하는 것이다.

upSet() 함수의 이웃해 생성은 크게 세 부분으로 이루어진다. 먼저, 새로 생성할 이웃해의 앞 부분인 1, 2, ..., $k-1$ 번째 위치에 대해서는 주어진 기존 해 sol과 동일한 작업들을 복사하며, 이는 <표 5>의 4~7번 줄에 나타나 있다. 다음으로, k번째 위치에 작업을 할당할 때는 제약조건을 위배하지 않는 후보해들 중, sol에서 k번째 위치를 점유하고 있는 작업 $sol[k]$를 제외하고, 남은 후보 항목 중 임의의 한 개를 선택하여 할당하며, 이는 <표 5>의 8~13번째 줄에 의해 실행된다. 마지막으로 생성할 이웃해의 뒤쪽 부분에 해당하는 $k+1$, $k+2$, ..., n번째 위치에 대해서는 sol과 같은 방식으로 후보를 선택하여 할당하되, 앞의 k번째 위치에서 해의 변형이 일어났기 때문에 앞 부분에서와 같이 sol의 항목들을 그대로 복사하는 것은 불가능하고, 할당 가능한 후보들 중, sol에서 가장 앞 부분을 점유하는 것을 선택하여 할당함으로써, 생성되는 이웃해의 후반부에서 sol과 유사한 특성이 나타나도록 한다. 이러한 과정은 <표 5>의 14~21번 줄에 의해 구현되었다.

<표 5> 후보순위 방식에 기반한 이웃해 생성(upSet)

```
1. Solution upSet(Solution sol, int k) {
2.          neighbor = null;
3.          items[i].isAssign = false;   i = 1, 2, ..., n
4.          for(j=1; j<k; i++) {
5.                      neighbor[j] = sol[j];
6.                      items[sol[j]].isAssign = true;
7.          }
8.          candidate_list = set of unassigned items;
9.          violate_list = set of unassigned items whose predecessors are not assigned
10.         candidate_list = candidate_list - violate_list - { sol[k] };
11.         next_item = a random element of candidate_list
12.         neighbor[k] = next_item;
13.         items[next_item].isAssign = true;
14.         for(j=k+1; j<=n; j++) {
15.                     candidate_list = set of unassigned items;
16.                     violate_list = set of unassigned items whose predecessors are not assigned;
17.                     candidate_list = candidate_list - violate_list;
18.                     next_item = the element of candidate_list with earliest order in sol;
19.                     neighbor[j] = next_item;
20.                     items[next_item].isAssign = true;
21.         }
22.         return neighbor;
23. }
```

<표 6>은 <표 1>의 24번 줄에서 볼 수 있는 타부리스트 갱신 과정으로, 매개변수로 받은 현재해와 다음해를 비교하여 해의 변형이 일어난 위치, 즉, <표 5>에서 k에 해당하는 위치를 upset_point로 정의한다. 예를 들어 현재 해가 [1 2 3 4 5]이고, 다음 해가 [1 2 4 3 5]인 경우, 3번째 위치에서 처음으로 현재 해와 다음 해 내용의 불일치가 일어나기 때문에 upset_point=3이 되고, 타부에는 (3, 4)가 저장된다.

<표 7>은 COTS의 다양화 전략을 의사코드로 나타낸 것으로, 기본적으로 초기해나 이웃해 생성 과정에서처럼 후보순위 방식에 의해 해를 생성하게 된다. 단, 초기해 생성 시 후보들 중 임의의 작업을 선택하고, 이웃해 생성 시 후보들 중 기존 해와 유사 또는 상이한 것을 선택했던 것과 달리, 다양화를 실시할 때는 장기 메모리를 참조하여, 이때까지 해의 후반부에 등장했던 작업들을 우선적으로 앞쪽에 배치, 새로운 해 공간으로 이동하는 것을 목적으로 한다.

Ⅵ. 단일기계 작업 순서결정 문제 적용 결과

4.1 실험 개요

본 연구에서는 한 대의 기계로 n개의 작업을 한 번에 한 개씩 순차적으로 처리해야 하는 단일기계 작업 순서결정 문제에 COTS를 적용하였으며, 작업들 간에 선행관계 제약이 있는 경우를 함께 고려함으로써, 첫째, 제약으로 인해 복잡도가 높아진 순서결정 문제를 대상으로

<표 6> 타부 목록 갱신(update_tabu_list)

```
1. void update_tabu_list(Solution s1, Solution s2) {
2.          upset_point = the earliest position k where s1[k] != s2[k];
3.          add_tabu(s1[upset_point], s2[upset_point]);
4. }
```

<표 7> 다양화(diversify)

```
1. Solution diversify(int[] order_sum) {
2.          item[i].isAssign = false, i = 1, 2, ..., n
3.          sol[i] = null, i = 1, 2, ..., n
4.          for(i=1; i<=n; i++) {
5.              candidate_list = set of unassigned items
6.             violate_list = set of unassigned items whose predecessors are not assigned
7.              candidate_list = candidate_list - violate_list
8.              next_item = the element of candidate_list with maximum order_sum[i];
9.              sol[i] = next_item
10.             item[next_item].isAssign = true
11.         }
12.         return sol;
13. }
```

COTS의 성능을 평가함과 동시에, 둘째, 선행관계 제약의 유무와 무관하게 COTSS가 순서결정 문제들을 범용적으로 처리할 수 있음을 보이고자 하였다.

실험에는 규모가 작은 경우(n=20)와 규모가 큰 경우(n=50)의 단일기계 작업 순서결정 문제를 사용하였으며, 선행관계 제약이 없는 경우부터 선행관계 제약을 늘려가면서 COTS를 사용한 풀이 결과를 관찰해보았다. 아울러, 성능 비교를 위해 전통적인 교환 방식으로 이웃해를 탐색하는 타부 서치 알고리즘을 함께 실험에 사용하여, 본 논문에서 제안한 COTS의 비교우위를 살펴보고자 하였다. 나아가, 선행관계 제약이 있는 경우에는 전통적인 타부 서치 알고리즘의 이웃해 과정에서 한 개 이상의 선행관계 제약을 위배하는 교환을 불허함으로써 실행불가능해가 생성되는 일을 방지하였다.

<표 8>은 작업 개수 n=20인 문제에서 주어지는 작업 각각의 처리에 소요되는 시간(processing time)을 나타내었다. 아울러, <표 9>~<표 11>은 작업 개수 n=20인 문제에 대하여 선행관계 제약 5개, 10개, 20개를 추가한 내역을 보여주며, 표들에서 선행 작업에 해당하는 번호의 작업이 반드시 후행 작업보다 일찍 처리되어야 함을 의미한다. 마찬가지로, <표 12>~<표 15>는 작업 개수 n=50인 문제와 관련된 정보들을 보여준다.

<표 8> 작업의 수 및 작업시간 (n=20)

Job	1	2	3	4	5	6	7	8	9	10
pt	24	8	11	13	12	35	29	6	16	9
Job	11	12	13	14	15	16	17	18	19	20
pt	37	30	14	15	26	24	17	19	21	22

<표 9> 선행제약 5개

선행 작업	1	2	11	11	10
후행 작업	2	8	9	10	8

<표 10> 선행제약 10개

선행 작업	1	2	11	11	10
후행 작업	2	8	9	10	8
선행 작업	7	19	17	20	13
후행 작업	19	17	2	17	8

<표 11> 선행제약 20개

선행 작업	1	2	11	11	10
후행 작업	2	8	9	10	8
선행 작업	7	19	17	20	13
후행 작업	19	17	2	17	8
선행 작업	18	9	5	3	16
후행 작업	9	5	3	10	4
선행 작업	15	6	6	12	11
후행 작업	1	1	10	17	12

<표 12> 작업의 수 및 작업시간 (n=50)

Job	1	2	3	4	5	6	7	8	9	10
pt	18	50	7	23	20	10	6	44	39	27
Job	11	12	13	14	15	16	17	18	19	20
pt	29	15	36	8	40	26	50	22	44	19
Job	21	22	23	24	25	26	27	28	29	30
pt	36	32	4	58	26	3	60	38	9	53
Job	31	32	33	34	35	36	37	38	39	40
pt	49	61	14	54	25	57	24	28	9	33
Job	41	42	43	44	45	46	47	48	49	50
pt	38	13	46	11	30	56	21	37	16	25

<표 13> 선행제약 10개

선행 작업	20	42	6	13	13
후행 작업	42	6	7	1	39
선행 작업	15	42	41	41	32
후행 작업	42	3	10	13	41

<표 14> 선행제약 25개

선행 작업	20	42	6	13	13
후행 작업	42	6	7	1	39
선행 작업	15	42	41	41	32
후행 작업	42	3	10	13	41
선행 작업	43	21	45	18	18
후행 작업	21	45	18	33	4
선행 작업	27	27	27	32	47
후행 작업	2	15	11	27	1
선행 작업	9	40	40	36	50
후행 작업	40	38	5	40	26

<표 15> 선행제약 50개

선행 작업	20	42	6	13	13
후행 작업	42	6	7	1	39
선행 작업	15	42	41	41	32
후행 작업	42	3	10	13	41
선행 작업	43	21	45	18	18
후행 작업	21	45	18	33	4
선행 작업	27	27	27	32	47
후행 작업	2	15	11	27	1
선행 작업	9	40	40	36	50
후행 작업	40	38	5	40	26
선행 작업	28	40	4	12	14
후행 작업	40	4	12	14	23
선행 작업	37	37	37	30	21
후행 작업	47	39	7	37	33
선행 작업	3	42	42	42	45
후행 작업	26	6	14	23	42
선행 작업	34	34	35	29	24
후행 작업	37	42	29	26	11
선행 작업	11	16	16	16	44
후행 작업	7	18	33	44	23

4.2 선행관계 제약 없는 경우

만약 선행관계 등과 같은 추가적인 제약조건이 없는 경우라면, 단일기계 작업 순서결정 문제의 정확한 최적해를 잘 알려진 SPT 규칙을 사용하여 구할 수 있다. 이에, 본 연구에서는 먼저, 제약조건이 없는 경우에 대해 COTS와 기존의 타부 서치(TS) 알고리즘을 적용하고, 산출된 결과를 SPT와 비교해보았다. 나아가, 실험에 필요한 매개변수들은 n=20인 문제에 대해 max_iteration=200, tabu_tenure=15, max_cold_period=15로 설정하고, n=50인 문제에 대해 max_iteration=1000, tabu_tenure=20, max_cold_period=50을 사용하였다. 나아가, 기존의 타부 서치와 COTS는 동일한 상황에 20회 반복 적용하여 산출된 해들의 목적함수 평균, 표준편차, 최대값 및 최소값을 집계해보았다.

<표 16>은 선행관계 제약이 없는 문제에 대한 비교실험 결과를 보여주고 있으며, n=20인 경우와 n=50인 경우 모두 기존 타부 서치의 목적함수 평균이 SPT 규칙을 통해 구한 최적해와 일치하고, 목적함수의 표준편차가 0임을 확인할 수 있다. 즉, 이는 기존의 타부 서치가 제약조건이 없는 문제에 대해 항상 정확한 최적해를 구했음을 보여준다. 반면, COTS의 경우, 그 차이가 크지는 않으나 평균적으로 정확한 최적해보다 미세하게 성능이 떨어지는 최적해를 구하는 경우가 많았으며, 목적함수의 표준편차도 0보다 큰 것으로 나타나, 매번 실행 시마다 얻어지는 해의 품질에 약간의 차이가 있었다.

4.3 선행관계 제약 있는 경우

한편, <표 17>, <표 18>에는 각 문제에 선행관계 제약을 추가하여 실험한 결과를 나타내었다. 이들을 보면, <표 16>에서와 달리, COTS를 이용하여 구한 해들의 목적함수의 평균이 기존 타부 서치에 비해 더 작아, 선행관계로 인해 복잡도가 높은 문제에 대해서는 COTS가 보다 우수한 성능을 나타내는 것을 볼 수 있다. 아울러, 목적함수의 표준편차 역시 COTS가 작게 나타나, 탐색 결과가 보다 안정적이다.

<표 16> 선행관계 제약 없는 경우 비교실험 결과

METHOD / VALUE	n=20			n=50		
	SPT	TS	COTS	SPT	TS	COTS
AVERAGE	3088	3088	3092.85	26343	26343	26349.3
STDEV		0	2.518		0	2.957
MAX		3088	3099		26343	26356
MIN		3088	3090		26343	26344

<표 17> 선행관계 제약 있는 경우 비교실험 결과 (n=20)

METHOD / VALUE	선행제약 5개		선행제약 10개		선행제약 20개	
	COTS	TS	COTS	TS	COTS	TS
AVERAGE	3591	3704.7	3693.9	3844.2	4182.7	4259
STDEV	70.218	129.512	50.192	86.161	32.9371	95.132
MAX	3735	3946	3805	4001	4233	4497
MIN	3477	3461	3628	3679	4109	4124

<표 18> 선행관계 제약 있는 경우 비교실험 결과 (n=50)

METHOD / VALUE	선행제약 5개		선행제약 10개		선행제약 20개	
	COTS	TS	COTS	TS	COTS	TS
AVERAGE	29289.7	30654.75	31622	35371.45	37064.25	40294.8
STDEV	368.34	1322.83	662.49	765.62	694.06	880.78
MAX	30190	34424	32981	36533	38387	41838
MIN	28698	29013	30709	33700	36035	38925

나아가, 알고리즘 실행에 소요되는 시간의 경우, 기존 타부 서치와 COTS 모두 n=20일 때 약 0.5~0.6초, n=50일 때 약 2~4초 가량으로 큰 차이가 없었으며, 이러한 결과를 볼 때, 본 논문에서 제안한 COTS가 제약조건의 유무에 상관없이 순서결정 문제를 유연하게 다룰 수 있고, 좋은 해를 산출해준다는 점을 확인할 수 있다.

Ⅴ. 결론 및 추후 연구과제

현대의 빠른 국제화로 인해 정보통신산업 뿐만 아니라 정밀기계공업의 발달은 점점 가속화되고 있다. 때문에 산업들의 발전은 점점 가속화 되어가며 제조업에서는 일정계획의 중요성이 계속 증가하고 있다. 또한 다품종 생산으로 인해 제조공정의 일정계획은 더 복잡해졌다. 하지만 많은 연구들로부터 좋은 일정계획방법들이 많음에도 불구하고 현실의 제조공정에 적용함에 있어서는 다음과 같은 한계점들이 존재한다. 먼저, 휴리스틱 방법들의 경우 비교적 적용이 단순하나 한 가지 문제에만 적용이 가능하고, 새로운 문제에 대한 해법을 개발하는 것이 어렵다. 메타 휴리스틱 방법의 경우 이러한 문제점이 어느 정도 완화되었으나, 여전히 추가적인 제약조건 등이 주어지는 경우를 유연하게 다루기 어렵다. 특히, 제조 현장의 일정계획 뿐만 아니라, 전자상거래, 대중교통 경로 관리 등에서 다양하게 활용되는 순서결정 문제의 경우, 이러한 단점들이 두드러진다.

이러한 문제를 해결하기 위해 본 논문에서는

유전 알고리즘에 적용되었던 후보순위 방식을 타부 서치에도 응용하여 COTS라는 새로운 탐색 방법을 설계 및 개발하였으며, 이를 단일기계 작업 순서결정 문제에 적용하여, COTS를 통해 제약조건이 있는 순서결정 문제에 대한 좋은 해를 신속히 탐색할 수 있음을 입증하였다. 특히, 기존에 유전 알고리즘과 타부 서치와 같은 방법들이 서로 독자적으로 연구되었던 것과 달리, 여러 가지 메타 휴리스틱 또는 탐색 기법에서 후보순위 방식이 범용적으로 활용될 수 있음을 보여, 다양한 해법의 개발에 토대를 마련하였으며, 높은 범용성 이외에도 후보순위 방식에 기반한 메타 휴리스틱 방법들의 경우 알고리즘의 수행 절차가 직관적이고 구현이 단순하다는 장점이 있어, 새로운 해법을 개발하는 데에도 폭넓게 이용될 수 있다. 아울러, 본 논문에서는 단일기계 작업 순서결정 문제만을 다루었으나, COTS는 조합최적화 중 순서결정 문제의 영역에 속하는 문제들에 범용적으로 적용할 수 있기 때문에, 향후 활용도가 매우 높을 것으로 기대된다.

끝으로, 저자들은 추후에도 알고리즘 및 성능 측면에서 시뮬레이티드 어닐링, 개미군집 알고리즘 등 다양한 메타 휴리스틱과의 비교실험을 통해 COTS의 성능평가를 진행할 것이며, 선행 제약뿐만 아니라 더 복잡한 형태의 제약조건을 갖는 문제들에도 적용해보아 후보순위 방식에 기반한 메타 휴리스틱 방법들이 갖는 장점을 탐구해나가고자 한다. 뿐만 아니라, COTS의 매개변수 최적화 및 순서결정 문제가 아닌 다른 조합최적화 문제에의 적용과 같은 부분 역시 중요한 추후 연구 과제들이다. 한편, 시스템 측면에서도 저자들은 향후 후보순위 방식의 해 탐색 기능을 소프트웨어 라이브러리로 구현하여 다양한 연구자들이 이를 통해 메타 휴리스틱 방법을 연구 및 개발하는 것을 지원하고자 한다.

참고문헌

김여근, 윤복식, 이상복, “메타휴리스틱,” 영지문화사, 1997.

Ahrned, I., and Fisher, W. W., “Due date assignment, job order release and sequencing interaction in job shop scheduling,” *Decision Sciences,* Vol23, No.3, 1992, pp.633-647.

Baker, K., “Introduction to sequencing and scheduling,” Wiley, 1974.

Chakhlevitch, K., Cowling, P., *“Hyperheuristics: recent developments”,* In: Adaptive and multilevel metaheuristics, Springer Berlin Heidelberg, 2008, pp.3-29.

Colin R. R., “Improving the efficiency of tabu search for machine sequencing problems,” *Journal of the Operational Research Society,* Vol.44, No.4, 1993, pp.375-382.

Cordreau, J. F., Laporte G., and Pasin, F., “Iterated tabu search for the car sequencing problem,” *European Journal of Operations Research,* Vol.191, No.3, 2008, pp.945-956.

Cordeau, J. F., and Maischberger, M., “A parallel iterated tabu search heuristic for vehicle routing problems,” *Computers and Operations Research,* Vol.39,

No.9, 2012, pp.2033-2050.

Gendreau, M., Hertz, A., and Laporte, G., "Tabu search heuristic for the vehicle routing problem," *Management Science,* Vol.40, No10, 1994, pp.1276-1290.

Gendreau, M., Laporte, G., and Semet, F., "A tabu search heuristic for the undirected selective traveling salesman problem," *European Journal of Operations Research,* Vol.106, No.2, 1998, pp.539-545.

Gao, J., Chen, R., and Deng, W., "An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem," *International Journal of Production Research,* Vol.51, No.3, 2013, pp.641-651.

Glover, F., "Tabu search: a tutorial," *Interfaces,* Vol.20, No.4, 1990, pp.74-94.

Hertz, A., "Tabu search for large scale timetabling problems," *European Journal of Operational Research,* Vol.54, No.1, 1991, pp.39-47.

Hopper, E. B. C. H., and Turton, B. C., "An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem", *European Journal of Operational Research,* Vol.129, No.1, 2001, pp.34-57.

Johnson, L. A., and Montgomery, D. C., *"Operations research in production planning, scheduling, and inventory control."* Wiley, 1974.

Kennedy, J., and Eberhaart, R., *"Particle swarm optimization,"* In: Proceedings of IEEE International Conference on Neural Networks, Vol.4, 1995, pp.1942 -1948.

Kim, J. W., "Developing a job shop scheduling system through integration of graphic user interface and genetic algorithm," *Multimedia Tools and Applications,* Vol.74, No.10, 2015, pp.3329-3343.

Kim, J. W., "Candidate order based genetic algorithm (COGA) for constrained sequencing problems", *International Journal of Industrial Engineering,* 2016, forthcoming.

Kirkpatrick, S., "Optimization by simulated annealing: quantitative studies," *Journal of Statistical Physics,* Vol.34, No.5-6, pp.975-986.

Koh, S. H. and Lim, D. J., "Single machine scheduling for minimizing earliness/ tardiness penalties," *Journal of Korea Management Engineers Society,* Vol.19, No.1 2014, pp.109-119.

Kovalyov, M., Musial, J., Urbanski, A., and Wojciechowski, A., "Internet shopping optimization problem," *International Journal of Applied Mathematics and Computer Science,* Vol.20, No.2, 2010, pp.385-390.

Kowalczyk, R., *"Constraint consistent genetic algorithms,"* In: IEEE International Conference on Evolutionary Computation, 1997, pp.343-348.

Laporte, G., "The vehicle routing problem: an overview of exact and approximate algorithms," *European Journal of*

Operational Research, Vol.59, No.3, 1992, pp.345-358.

Lenstra, J. K., *"Local search in combinatorial optimization,"* Princeton University Press, 1997.

Lü, Z., and Hao, J. K., "Adaptive tabu search for course timetabling," *European Journal of Operations Research,* Vol.200, No/1, 2010, pp.235-244.

Lawler, E. L., "Sequencing jobs to minimize total weighted completion time subject to precedence constraints," *Annals of Discrete Mathematics,* Vol.2, 1978, pp.75-90.

Lee, K. S., and Geem, Z. W., "A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice," *Computer Methods in Applied Mechanics and Engineering,* Vol.194, No.36, pp.3902-3933.

Lenstra, J. K., and Kan, A. R., *"Complexity of scheduling under precedence constrains,"* In: International Conference on Knowledge based Intelligent Electronics Systems, Vol.2, 1978, pp.60-66.

López-Locés, M. C., Rege, K., Pecero, J. E., Bouvry, P., and Huacuja, H. J. F., *"Trajectory metaheuristics for the internet shopping optimization problem,"* In: Design of Intelligent Systems Based on Fuzzy Logic, Neural Networks and Nature-Inspired Optimization. Springer International Publishing, 2015, pp.527-536.

Meeran, S., and Morshed, M. S., "A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study," *Journal of Intelligent Manufacturing,* Vol.23, No.4, 2012, pp.1063-1078.

Misevičius, A., "Using iterated tabu search for the traveling salesman problem," *Information Technology and Control,* Vol.32, No.3, 2015, pp.29-40.

Nowicki, E., and Smutnicki, C., "A fast tabu search algorithm for the permutation flow-shop problem," *European Journal of Operational Research,* Vol.91, No.1, 1996, pp.160-175.

Pezzella, F., and Merelli, E., "A tabu search method guided by shifting bottleneck for the job shop scheduling problem," *European Journal of Operational Research,* Vol.120, No.2, 2000, pp.297-310.

Sule, D. R., *"Industrial Scheduling,"* PWS publishing, 1997.

Toth, P., and Vigo, D., "The granular tabu search and its application to the vehicle-routing problem," *Informs Journal on Computing,* Vol.15, No.4, 2003, pp.333-346.

정성욱(Jeong, Sung-Wook)

2014년 동아대학교에서 산업경영공학 학사학위를 취득하였고 현재 동아대학교 산업경영공학과 석사과정에 재학중이다. 주요 관심분야는 생산 및 운영관리, 메타 휴리스틱 알고리즘 등이다.

김준우(Kim, Jun-Woo)

2009년 KAIST 산업 및 시스템공학과에서 박사학위를 취득하였고 2011년부터 현재까지 동아대학교 산업경영공학과 조교수로 재직 중이다. 주요 관심분야는 데이터마이닝, 인공지능, 지능형 시스템, 조합최적화 및 메타 휴리스틱 알고리즘 등이다.

<Abstract>

Solving the Constrained Job Sequencing Problem using Candidate Order based Tabu Search

Sung-Wook Jeong · Jun-Woo Kim

Purpose

This paper aims to develop a novel tabu search algorithm for solving the sequencing problems with precedence constraints. Due to constraints, the traditional meta heuristic methods can generate infeasible solutions during search procedure, which must be carefully dealt with. On the contrary, the candidate order based tabu search (COTS) is based on a novel neighborhood structure that guarantees the feasibility of solutions, and can dealt with a wide range of sequencing problems in flexible manner.

Design/methodology/approach

Candidate order scheme is a strategy for constructing a feasible sequence by iteratively appending an item at a time, and it has been successfully applied to genetic algorithm. The primary benefit of the candidate order scheme is that it can effectively deal with the additional constraints of sequencing problems and always generates the feasible solutions. In this paper, the candidate order scheme is used to design the neighborhood structure, tabu list and diversification operation of tabu search.

Findings

The COTS has been applied to the single machine job sequencing problems, and we can see that COTS can find the good solutions whether additional constraints exist or not. Especially, the experiment results reveal that the COTS is a promising approach for solving the sequencing problems with precedence constraints. In addition, the operations of COTS are intuitive and easy to understand, and it is expected that this paper will provide useful insights into the sequencing problems to the practitioners.

Keywords: job sequencing problem, precedence constraint, combinatorial optimization, meta heuristic, tabu search

* 이 논문은 2015년 11월 27일 접수, 2016년 2월 12일 1차 심사, 2016년 3월 16일 게재 확정되었습니다.