「정보시스템연구」 제28권 제4호
한국정보시스템학회
2019년 12월, pp. 175～197

http://dx.doi.org/10.5859/KAIS.2019.28.4.175

시간-자원 트레이드오프 프로젝트 스케줄링 문제 해결을 위한 시뮬레이티드 어닐링 기반 휴리스틱 알고리즘 개발

김 건 아* · 서 윤 호**

〈목 차〉

Ⅰ. 서 론

제한된 자원의 양과 선행 제약을 가진 활동들을 고려해 전체 완료 시간을 최소화시키는 프로젝트 스케줄링 문제인 Resource Constrained Project Scheduling Problem (RCPSP)은 Kelly(1963)에 의해 처음 소개되었다. 그 이후 Blazewicz et el(1983)에 의해 NP-Hard 문제로 증명된 후, 다양한 Heuristic Algorithm들이 개발되었다. Christofides et el(1987)이 Branch and Bound Algorithm을 Bell and Park(1990)이 A* Algorithm을 적용하였고, Icmeli and Erenguc(1994)가 Tabu Search를 Jia and Seo(2013a)가 Particle Swarm Optimization을 적용하였다. Kolisch(1995)가 기존의 단일 자원에서 고려되던 RCPSP에서 파생된 다중 자원을 고려한 Multi-Mode RCPSP(MRCPSP)를 연구하였다. 하지만, 기존의 RCPSP에서는 각 활동들의 작업 기간과 투입될 비용(자원)의 양이 고정되어 있다는 한계가 존재하였다. 실제 프로젝트에서는 투입되는 비용(자원)에 따라 전체 작업 시간이 단축 될 수 있기 때문에, 보다 현실적인 스케줄링 문제를 해결하기 위해서 시간과 비용에 상관 문제를 고려한 Discrete Time/Cost Trade-off Problem(DTCTP)과 시간과 자원의 상관 문제를 고려한 Discrete Time/Resource Trade-off Problem(DTRTP)가 연구되고 있다.

* 고려대학교 산업경영공학과, tomis0219@korea.ac.kr(주저자)
** 고려대학교 산업경영공학과, yoonhoseo@korea.ac.kr(교신저자)

본 연구에서는 전체 완료 시간(makespan)을 최소화하기 위해 자원과 시간의 관계를 고려한 문제인 Discrete Time/Resource Trade-off Problem(DTRTP)에 대해 다룬다. 기존의 연구들은 자원과, 시간으로 구성된 유한한 조합들 중 전체 makespan을 최소화하게 만드는 최적 조합들을 찾아낸다. 하지만 기존의 연구는 시간과 자원을 정수 내에서만 고려하였기 때문에, 현실적인 계획에 적용되는데 한계가 존재했다. 예를 들어 정수의 시간만을 고려한다면 년, 월, 일 단위의 계획은 세울 수 있지만, 시간이나 분 단위의 계획까지 통합하여 계획을 세우기에는 어려움이 존재한다. 또한 인력의 숙련도를 고려해야 하는 경우 정수의 자원으로는 표현하기 어렵다. 이를 해결하기 위해 실수의 시간이나 자원을 고려한 스케줄링에 대한 연구가 수행되어 왔지만, 시간과 자원의 관계를 동시에 고려한 문제들에 대해서는 연구가 되지 않았다. 기존의 연구된 정수 기반의 알고리즘은 유한한 조합 내에서 최적 조합을 도출하기 위해 개발되었다. 하지만 시간과 자원을 실수 차원에서 고려할 경우 무한한 조합이 발생하게 된다. 이로 인해 기존의 알고리즘을 적용하기에는 어려움이 있다. 이에 본 연구에서는 실수의 시간과 자원 관계를 고려한 DTRTP에서 최적 조합을 도출해내는 휴리스틱 알고리즘에 대한 연구를 수행했다.

Ⅱ. 기존 연구

De Reyck et el(1998)에 의해 처음 소개된 Discrete Time/Resource Trade-off Problem (DTRTP)은 투입되는 자원의 양이 증가함에 따라 작업 시간을 단축할 수 있다는 점을 고려하여, 유한한 (시간, 자원) 조합 내에서 전체 완료 시간을 최소화하는 (시간, 자원)의 최적 조합을 찾는 프로젝트 스케줄링 문제이다. Demeulemeester et el(2000)에 의해서 NP-Hard 문제임이 밝혀진 이후, 이를 해결하기 위한 Algorithm들이 제안되어 왔다. Ranjbar and Kianfur(2007)가 Genetic Algorithm을 기반으로 한 방법을 제안하였고, Ranjbar et el(2009)은 Scatter Search를 통해 DTRTP를 해결 했다. 기존의 DTRTP 연구에서는 시간과 자원을 정수 값으로 제한하고, 유한한 시간과 자원의 조합을 최적화해 전체 완료시간을 최소화하는 문제만을 다루고 있다. 하지만 현실에서의 스케줄링 문제에서는 투입되는 시간과 자원이 실수 값으로 고려되어야 하는 필요성이 존재한다.

실수 값으로 시간과 자원을 고려해야 하는 이유는 시간의 경우 1일 단위는 정수 값으로 표현할 수 있으나, 1일을 시간, 분, 초 단위로 나눠서 작업을 배치해야 하는 경우 실수로 표현해야 한다. 예를 들어, 실시간 작업 배치계획(Hong and Leung, 1988), 항공기 착륙시간 계획(Ciesielski and Scerri, 1998)에서는 시간을 실수로 표현하여 스케줄링 해야 한다. 인력 자원을 할당하는 문제의 경우에는 각 활동에 필요한 인력을 배치하는 것이 중요하다. 하지만 숙련도를 고려하지 않고 계획을 생성할 경우에는 실제 작업 기간과 차이가 발생하게 된다. Dewi and Septiana(2015)는 육체적 정신적인 부문을 세분화하여 각 작업에 숙련도를 체크하고 능숙한 작업자들을 1이상의 실수로 표현하여 인력(자원) 배치 계획을 도출 하였다. 이와

같은 이유로 현실에 가까운 작업 계획을 도출하기 위해 시간과 자원을 실수 값으로 고려해야 할 필요성이 존재한다. 하지만 기존의 DTRTP에서는 이러한 실수 값의 시간과 자원을 고려하지 않았다. 실수 값의 시간과 자원을 고려할 경우, 투입되는 자원에 따라 조정되는 시간의 관계에 대한 조합이 무한히 발생하게 되는데, 기존의 연구에서는 유한한 (시간, 자원)의 조합을 최적화하는 문제만을 다루고 있어서 기존의 연구들에서 제안된 방법으로는 실수 값을 가지는 무한한 (시간, 자원) 조합에서 최적 조합을 도출하기 어렵다. 이를 해결하기 위해 새로운 Heuristic Algorithm을 제안했다.

본 연구에서 제안한 Heuristic Algorithm은 초기 계획에 최소 자원을 활동들에 배치했다. 그 이후 초기 계획을 기반으로 makespan을 최소화하기 위해 투입 가능한 자원을 각 활동에 투입하여 시간을 단축하는 과정을 통해 무한한 조합 내에서 최적의 (시간, 자원) 조합을 도출하였다. 기존의 방식처럼 유한한 개수의 (시간, 자원) 조합을 각 일정 계획에 대입해보는 방식이 아니라, 배치된 활동의 위치에서 자원을 추가로 투입하여 makespan을 최소화하고, (시간, 자원)의 최적 조합을 도출하는 Algorithm이다. 본 연구에서 고려하는 실수 기반의 DTRTP에 대해서 3절에서 정의하고, 4절에서는 실수 기반에서 시간과 자원의 최적 조합을 도출하는 Simulated Annealing 기반 Heuristic Algorithm을 제안했다. 5절에서는 제안된 Algorithm의 효율성을 입증하기 위해 기존의 DTRTP에서 적용된 Genetic Algorithm 기반 방법론 (Ranjbar, 2007)과 비교하는 실험을 진행했으며, 마지막으로 연구 결론에서 대한 부분을 6절에서 서술했다.

Ⅲ. 문제 정의

DTRTP의 목적은 자원 투입에 따라 조정되는 시간의 조합 중 최적의 조합을 찾아 makespan을 최소화 하는 것이다. 각 활동들은 투입된 자원(r_i)에 따라 조정된 작업 시간(d_i)을 가지며, (d_i,r_i)의 조합 형태로 나타내며, 계획에 배치된 활동들은 작업 시작 시간(s_i)과 작업 완료 시간(f_i)을 가진다. 자원의 경우 하나의 재사용 가능한 자원이라 가정한다. 문제 정의를 위해 아래와 같은 기호와 변수를 정의했다.

기호와 변수

$i = \{1,2,...,n\}$	Set of activities with index i
$l = \{1,2,....,L\}$	Set of solution with index l
t	time t on a real number; $t \geq 0$
R	Resource limit at time t
P_i	set of predecessor activities of activity i
$s_i = \{s_1,....,s_n\}$	Set of activity start time in scheduling
$f_i = \{f_1,....,f_n\}$	Set of activity finish time in scheduling
$r_i = \{r_1,....,r_n\}$	Set of activity resource in scheduling
$d_i = \{d_1,....,d_n\}$	Set of activity duration in scheduling;
$nr_i = \{nr_1,....,nr_n\}$	Set of activity normal resource
$nd_i = \{nd_1,....,nd_n\}$	Set of activity normal duration
$cr_i = \{cr_1,....,cr_n\}$	Set of activity crash resource
$cd_i = \{cd_1,....,cd_n\}$	Set of activity crash duration
$k_i = \{k_1,....,k_n\}$	Set of add resource per a day

본 연구에서 DTRTP의 목적함수와 제약식은 Kolisch and Hartmann(1999)의 수리모형을 참

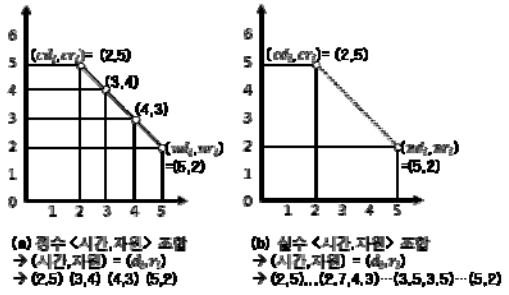
조한다. 참조된 DTRTP의 목적함수와 제약식은 자원 제약 하의 프로젝트 스케줄링 문제의 기본적인 목적식과 제약식으로 본 연구의 문제와 공통된 부분이다. 공통된 목적함수는 makespan을 최소화시키는 목적함수이며, 공통된 제약 조건으로는 선행 제약과 자원 제약이 존재하는데, 선행 제약 조건은 j 활동의 선행 활동 i는 P_j에 포함되며, 활동 j의 시작시간(s_j)보다 선행 활동 i의 완료시간(f_i)이 작거나 같아야 한다. 또한 각 시간 t에 배치된 활동들의 투입된 자원의 합은 자원 한도(R)를 넘을 수 없는 자원 제약 조건을 만족해야한다.

기존의 DTRTP 연구에서는 정수 기반의 유한한 (시간, 자원)의 조합(De Reyck et el, 1998)을 최적화하여 makespan을 최소화시키는 목적함수를 만족시키기 위해 Heuristic Algorithm들을 개발하였다. 그러나 본 연구에서는 실수 기반의 무한한 (시간, 자원)을 최적화해야 되기 때문에 새로운 Heuristic Algorithm이 필요하다. 실수 기반의 Heuristic Algorithm을 위해 계획에 배치된 각 활동들의 작업 시간(d_i)과 투입된 자원(r_i)간의 관계의 선형성이 존재한다고 가정하였다. 시간과 자원 간의 선형성이 존재함에 따라 다음과 같은 식이 존재한다. 투입되는 자원에 따라 시간의 조정이 선형적으로 발생하는 식 (1)과 시간의 조정에 따라 투입될 자원 또한 선형적으로 변화하는 식 (2)가 존재한다. 시간과 자원 간의 관계가 선형적이라는 가정하의 i활동에 작업 시간의 변화에 따라 투입되는 자원의 변화 값인 k_i가 존재한다. k_i는 활동 i의 최대 투입 가능 자원(cr_i)에서 최소 투입 가능 자원(nr_i)에 차이 값에 최대 작업 기간(nd_i)와 최소 작업 시간(cd_i)의 차이 값을 나눈 값이다 k_i를 통해 활동 i에 투입된 자원의 양(r_i)에 따라 조정되는 작업 시간(d_i)과 조정된 작업 시간(d_i)에 따라 투입되어야 할 자원의 양(r_i)이 계산된다.

$$d_i = nd_i - \frac{r_i - nr_i}{k_i} \qquad nd_i \geq d_i \geq cd_i \qquad (1)$$

$$r_i = nr_i + (nd_i - d_i) \times k_i \quad cr_i \geq r_i \geq nr_i \qquad (2)$$

또한 정수 기반의 시간, 자원을 고려하는 것이 아닌, 실수 기반의 시간, 자원을 고려하기 때문에 변수들의 범위가 의미하는 것이 다르다. 예를 들어 t(시간)의 범위인 $t \geq 0$은 0보다 큰 실수 값을 의미한다. 또한 활동 i의 d_i, r_i를 계산하는 식 (1), (2)에서 또한 d_i, r_i 값이 해당 기간에 포함된 실수 값을 의미한다. 이러한 범위의 의미 차이는 시간, 자원의 조합 구성에 있어 다음과 같은 차이를 만들어낸다.



<그림 1> 시간과 자원의 조합

이전의 연구에서는 <그림 1>의 (a)처럼 정수 기반의 유한한 시간-자원의 조합을 활동에 할당하며 최적 조합을 도출한다. 하지만 본 연구에서는 실수 기반의 시간-자원의 조합들을 고려해야 하기 때문에 <그림 1>의 (b)와 같이 무한한 조합들이 존재한다. 그러므로 기존의 방법

대로 실수 값으로 구성된 무한한 조합들을 활동에 할당하는 방식으로는 유효 시간 내의 최적의 조합을 구하기 어렵다. 본 연구에서는 무한한 시간과 자원의 조합들에서 최적의 조합을 구하기 위해 개발된 Heuristic Algorithm에 대해서 다음 장에서 소개한다.

Ⅳ. Heuristic Algorithm

본 연구에서 제안하는 Heuristic Algorithm은 각 활동들이 가지는 무한한 (시간, 자원)의 조합에서 makespan을 최소화하게 만드는 최적 조합을 찾아 일정 계획을 생성하는 Algorithm이다. 해당 Algorithm은 Sub Group Solution과 Push Algorithm, Simulated Annealing으로 구성된다. Sub Group Solution은 전체 활동들 중 특정 시간(t)에 동시 작업할 수 있는 활동들을 찾아 Sub Group으로 묶어 배치 순서로 표현하는 방식이다. 해당 방식은 각 Sub Group 내에 포함된 활동들이 배치된 Time Table에서 자원의 사용율을 최대한 높여 완료시간을 단축하기 위해서 사용된다. Sub Group들의 활동들은 Push Algorithm을 통해 makespan을 최소화하기 위해 최대한의 자원을 사용하며 배치되게 된다. Push Algorithm은 각 Sub Group들의 작업 기간을 최소화하게 만드는 각 활동들의 최적 조합(d_i,r_i)을 도출하기 위해 개발한 방법이다. Push Algorithm을 통해 최적의 조합을 도출하기 위해서는 Sub Group Solution으로 가능한 모든 Solution을 탐색해야 한다. n개의 활동들을 g개의 Sub Group들로 묶어서 Solution을 구성했기 때문에 시간복잡도는 n!보다 단축되었다. 하지만 활동의 개수가 늘어날수록 Sub Group의 개수는 증가하게 되고 높은 시간복잡도인 g!를 가지게 된다. 이에 따라 많은 활동을 배치할 때도 유효한 시간 내에 좋은 결과를 찾기 위해서 Simulated Annealing을 사용했다.

4.1 Sub Group Solution

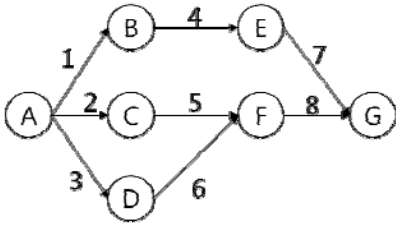
기존의 연구에서는 활동들의 배치 순서만을 표현한 Permutation Solution 구조가 사용되었다. 하지만 본 연구에서 사용한 Sub Group Solution은 n개의 활동들을 g개의 Sub Group으로 묶어서 Solution을 구성한다. 동시에 작업이 가능한 활동들을 Sub Group으로 묶어서 Solution으로 표현하는데, 그 이유는 계획에 배치된 활동들에 가능한 최대 자원을 투입하여 makespan을 단축하는데 목적이 있다.

makespan을 최대한 단축하기 위해서는 각 기간 동안 배치 된 활동들이 최대 자원 한도(R)에 가깝게 자원을 사용해야 한다. 배치된 활동들이 최대 자원 한도(R)에 가깝게 자원을 사용하는 경우는 하나의 활동 혹은 동시에 작업되는 활동들이 최대 자원 한도(R)에 가깝게 자원을 사용하는 경우이다. 이를 위해 제안된 Sub Group Solution에서는 기존의 Solution처럼 활동의 배치 순서 리스트만을 의미하는 것이 아니라, 동시에 배치되는 활동들에 대해 고려한다. 동시 작업이 가능한 활동이란 배치가 안 된 활동들 중 모든 선행 활동들이 앞서 배치가 되어, 현재 배치가 가능한 활동들을 의미한다. <그림 2>를 통해 예를 들자면, 모든 활동이 배치가 안 되어 있는 경우, {1,2,3}이 동시 작업이 가능한 활동들이 된다. {1,2}가 배치되어 있는

경우, {3,4,5}가 동시 작업이 가능한 활동들이다.

<표 1> Job8의 Job Data

i	nd_i	nr_i	P_i	cd_i	cr_i	k_i
1	4	2	-	2	5	1.5
2	3	1	-	2	3	2
3	2	1	-	2	1	0
4	6	4	1	4	6	1
5	10	2	2	7	4	0.66
6	8	1	3	6	3	1
7	9	3	4	5	6	0.75
8	2	1	5,6	2	1	0



<그림 2> Job8의 AOA Network

Sub Group 내의 각 활동들은 makespan을 최소화하기 위해서 최대한의 자원이 투입되어진다. 이를 위해 각 활동들은 초기 값으로 가장 적은 자원이 투입된 (nd_i ,nr_i)가 할당된다. 각 Sub Group으로 구성되는 활동들은 최소 투입 자원들의 합($\sum nr_i$)이 자원한도(R) 이하로 구성이 되게 된다. 그 이유는 투입 가능한 자원이 자원한도(R)를 초과하여 투입될 수 없기 때문이다. 이를 위해 각 Sub Group의 구성 시 최소 투입 가능한 자원($\sum nr_i$)을 고려한다. Sub Group Solution에서 각 Sub Group을 생성하기 위해서는 다음 Step이 필요하다

Step1. 활동(1,2,..,n)들의 중 배치된 활동(JobOrder[])들을 확인해, 동시에 작업이 가능한 활동(Selectable[])들을 찾는다.

Step2. 동시에 작업이 가능한 활동(Selectable[])들 중 하나의 활동을 선택한다.

Step3. 선택된 활동의 최소 투입 가능 자원(nr_i)을 Using Resource에 합한다.

Step4. Using Resource가 R을 초과하지 않는다면, Sub[]과 JobOrder[]에 추가하고 Step2로 돌아가라. 그러나 Using Resource가 R을 초과하면 Sub[]을 완성하고 다음 Sub[]를 생성하기 위해 Step1으로 돌아가라. 전체 작업이 배치되었다면 Step5로 간다.

Step5. Sub[]에 저장된 Sub Group들을 하나의 Solution으로 저장한다.

1에서 5의 Step를 바탕으로 구성된 Sub Gorup들은 하나의 Sub Group Solution으로 구성되며, <표 2>와 같이 표현된다.

<표 2> Sub Group Solution

Solution	Sub Group Sequence
sol_l	{1,2,3} {4,5} {6,7} {8}

4.2 Push Algorithm

Push Algorithm은 각 Sub Group 내에 속한 활동들의 작업 기간을 단축시킴으로써 전체 makespan을 최소화하는 방법이다. Push Algorithm을 위해서는 다음과 같은 기호와 변수가 추가로 사용된다.

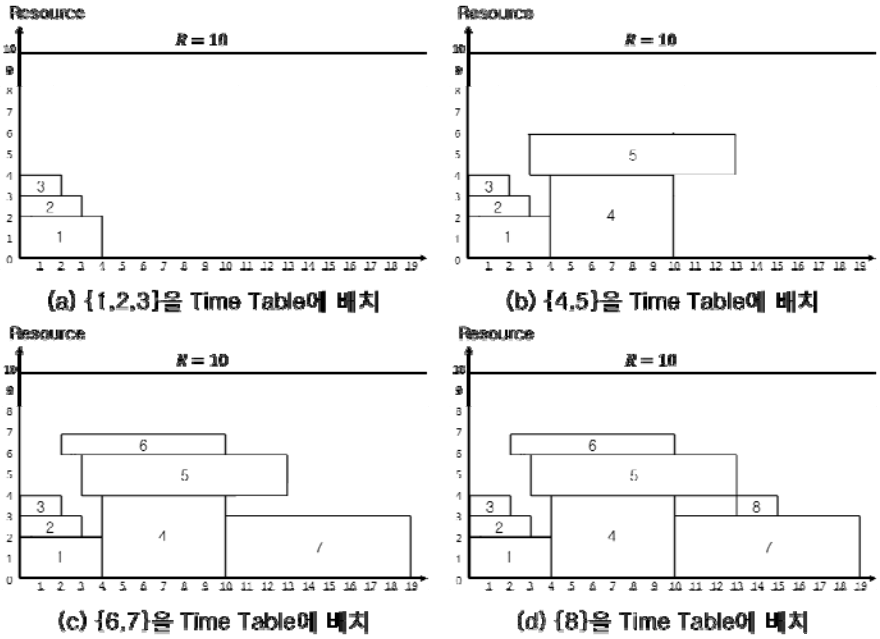
기호와 변수

e = {1,2,..,E}	Set of Related groups with index e
rel_e = {rel_1,rel_2,...., rel_E}	Set of Related group
ar_i	Available resource of activity i
ad_i	Available duration of activity i

작업 기간을 단축하기 위해서는 자원을 추가로 투입하여야 한다. 추가로 투입 가능한 자원(ar_i)을 확인하기 위해서는 이전에 Push되어 배치가 완료된 Sub Group들을 확인해야 한다. 이는 배치될 활동들과 배치된 활동들의 일정 계획 위치에 따라 배치할 활동 i의 배치 구간에서 사용된 자원의 양이 다르기 때문이다. 즉, 이전에 배치가 완료된 활동들과 새로 배치될 활동들의 위치를 비교하는 과정을 통해 사용된 자원을 확인하고, 배치할 각 활동들의 투입 가능한 자원의 양(ar_i)을 확인한다. 그 후 자원을 투입해 작업기간을 단축하는 Push 과정을 거쳐 makespan을 최소화하는 (시간, 자원)의 최적 조합을 도출하게 된다.

4.2.1 초기 배치

Sprecher et el(1995)가 제안한 스케줄링 방법인 SSGS(Serial Schedule Generation Schemes)는 DTRTP에서 (시간, 자원)의 조합들을 Time Table에 배치하는 데 효과적임이 증명되었다(De Reyck et el, 1998). Time Table에 배치되는 활동들은 초기 값으로 (nd_i ,nr_i)을 가지며, Sub Group 내의 활동들의 순서대로 배치된다. 이 때, 선행 제약과 자원 제한에 위반되지 않는 실행 가능한 계획으로 생성된다. <표 2>의 Solution인 {1,2,3} {4,5} {6,7} {8}을



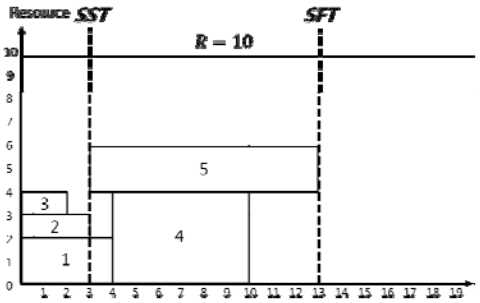
<그림 3> SSGS를 통한 {1,2,3} {4,5} {6,7} {8} 계획 생성

Sub Group 순서대로 SSGS를 통해 계획을 생성하게 되면, 위의 <그림 3>과 같이 배치되게 된다.

4.2.2 기존 배치 활동 확인

Time Table에 배치된 Sub Group들의 각 활동들은 초기 값으로 (nd_i ,nr_i) 가지게 되는데, 해당 초기 값은 가장 적은 자원을 투입했기 때문에 가장 긴 작업시간을 가지게 된다. 전체 makespan을 단축하기 위해서는 각 Sub Group들의 작업시간을 단축해야 하며, 이를 위해서 배치할 활동들에 투입 가능한 자원의 양을 확인해야한다. 자원의 양을 확인하기 위해 배치된 Sub Group의 시작 시간(Sub group Start Time : SST)과 완료 시간(Sub group Finish Time : SFT)을 확인하고, 해당 기간에 포함된 활동들을 확인한다. 해당 기간에 포함된 활동들은 이전의 배치가 완료된 활동들과 새롭게 배치될 Sub Group의 활동들로 나뉘게 된다.

<그림 3>의 (b)를 예를 들어 <그림 4>에서 이를 확인한다면, Sub Group {4,5}의 SST는 3, SFT는 13이다. 해당하는 기간 동안 배치가 완료된 활동은 {1}이며, 새롭게 배치될 활동들은 {4,5}이다. 2가지로 분류하는 이유는 단축할 활동들({4,5})을 확인하고, 각 활동들에 투입 가능한 자원의 양을 Push 실행 과정에서 확인하기 위해서이다. 배치될 각 활동들에 투입 가능한 자원(ar_i)은 $nr_i \leq ar_i \leq R$의 조건과 SST와 SFT 사이에 배치된 활동들이 사용한 자원에 의해 결정된다. 예를 들어 <그림 4>에서 활동 4의 투입 가능한 자원은 $4 \leq ar_4 \leq 10$의 조건을 만족해야 한다. 활동 4가 배치된 기간에 함께 배치되어 있는 활동은 5뿐이므로, 자원 한도(R) 10에서 활동 5에 투입된 자원인 2를 제외한 8이 최대 투입 가능 자원이 된다. 하지만 활동 4에 최대 투입 자원(cr_4)은 8보다 작은 6이므로 활동 4의 투입 가능 자원은 $4 \leq ar_4 \leq 6$이 된다.



<그림 4> Sub Group {4,5}의 SST와 SFT

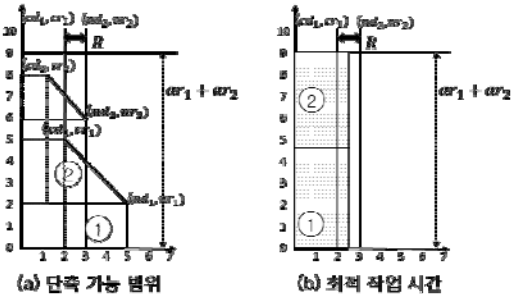
4.2.3 Push Algorithm 설계

앞 장에서 Sub Group들의 배치와 함께 배치가 완료된 활동들을 확인 했다. 배치된 활동들을 통해 투입 가능한 자원을 확인한 후, 배치된 Sub Group의 작업기간을 단축하기 위해서 Push를 실행한다. Push는 SSGS로 배치된 Sub Group의 활동들에 할당된 (시간, 자원)을 조정해 단축하게 된다. Push는 배치할 활동들 간의 단축 가능 기간을 확인하여 Related Group을 생성하는 단계, Related Group들의 makespan을 최소화하게 하는 각 활동들의 최적 작업 시간(d^*)를 찾는 과정을 실행한다.

(1) Related Group 생성

Related Group이란, Sub Group에 포함된 활동들의 단축 가능 범위를 확인하고, 해당 범위

가 겹치는 활동들 간의 최적 작업 시간(d^*)을 고려하기 위해 묶은 Group을 의미한다. 단축 가능 범위란 각 활동의 자원과 시간 조합의 범위인 $(nd_i, nr_i)\cdots(cd_i, cr_i)$을 의미하며, 최적 작업 시간($d^*$)란 Related Group에 포함된 활동들의 makespan을 최소화하는 시간을 말한다. Related Group을 생성하는 이유는 활동들 간의 단축 가능 범위가 겹치는 경우, 해당 활동들의 makespan을 최소화 하는 시간(d^*)이 겹치는 범위에 존재하기 때문이다. 이는 다음 단축 가능 범위가 겹치는 1, 2활동의 관계를 통해 설명한다.



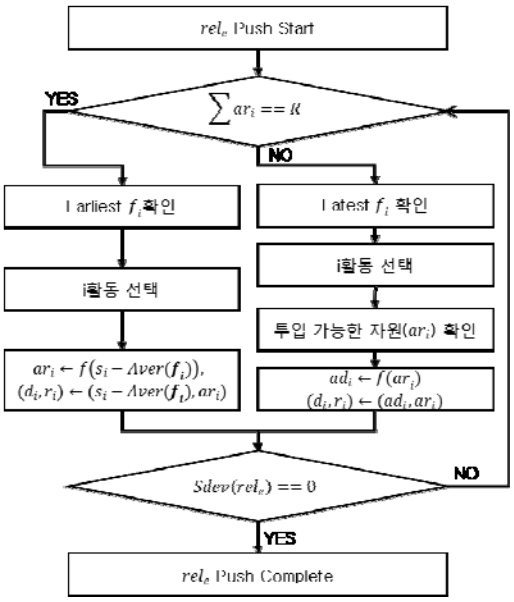
<그림 5> 활동 1,2의 단축 가능 범위와 최적 작업 시간

<그림 5>에서 활동 1의 단축 가능 범위는 (5,2)…(2,5) 사이의 실수 값을 가지는 (시간, 자원)의 조합이며, 활동 2의 단축 가능 범위 또한 (3,4)…(1,6) 사이의 (시간, 자원)의 조합이다. 해당 활동 1,2의 투입 가능한 자원의 합은 $nr_1 + nr_2 \le ar_1 + ar_2 \le R$의 조건을 만족하는 $ar_1 + ar_2$이다. 즉, $3 \le ar_1 + ar_2 \le 9$의 조건을 만족해야 한다. 이 두 활동들의 작업 시간을 최소화하기 위해서는 투입할 수 있는 $ar_1 + ar_2$의 최대 자원인 9를 투입해 작업 시간을 결정해야 한다. 이때 최적 작업 시간(d^*)은 <그림 5>의 (b)와 같이 1,2 활동이 동시에 끝나는 것이다. 동시에 끝나기 위해서는 1, 2활동의 단축 가능 범위가 겹쳐야 한다. 이를 위해 두 개 이상의 활동을 포함하는 Related Group(rel_e)을 생성한다. 단축 가능한 범위가 겹치지 않는 경우에는 고려하지 않아도 되기 때문에 홀로 Related Group을 생성한다. Related Group을 생성 한 후 각 Related Group의 makespan을 최소화하는 각 활동의 최적 작업 시간(d^*)를 계산한다.

(2) d* 탐색

Push는 활동들의 전체 makespan을 최소화하는 최적의 (시간, 자원) 조합을 찾는 방법이다. 최적의 조합을 도출해내기 위해서는 모든 조합을 확인해야 하지만, 실수 값의 무한한 (시간, 자원)의 조합을 가지고 있기 때문에 모든 경우의 수를 고려하기 위해서는 무한한 시간이 걸리게 된다. 이를 해결하기 위해 각 활동의 최적 작업 시간(d^*)을 도출하기 위한 Push 과정<그림 6>을 제안한다.

Push 과정은 자원 제한을 고려하며, 각 활동들의 자원을 투입하고 빼면서 전체 makespan이 최소화되는 조합을 찾는다. <그림 6>은 Related Group(rel_e)의 활동들에 대해 Push 과정을 실행한다. 가장 먼저 Push할 활동들에 투입 가능한 여유 자원이 있는지 확인한다. 여유 자원이 있다면, 가장 늦게 끝나는 활동 i를 찾아서 가능한 최대한의 자원(ar_i)을 투입한다. 투입된 자원(ar_i)에 따른 변경된 작업 시간(ad_i)은 식 (1)을 통해 계산한다. 계산을 통해



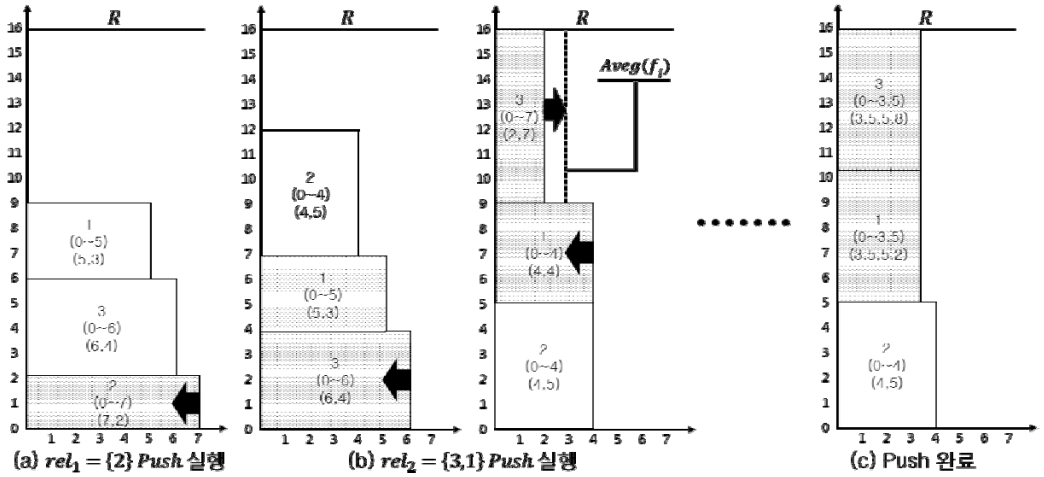
<그림 6> Push Flow Chart

나온 조합(ad_i, ar_i)을 활동의 계획정보(d_i, r_i)에 업데이트한다. 여유 자원이 없을 경우, 가장 일찍 끝나는 활동 i의 작업 시간(d_i)이 rel_e의 포함된 활동들의 평균 f_i에 끝나도록 작업 시간을 늘리고 추가된 작업 시간$(s_i - Aver(f_i) = ad_i)$에 따른 필요한 자원(ar_i)을 식 (2)를 통해 계산한다. 조정된 조합(ad_i, ar_i)을 계획 정보에 업데이트 한다. 해당 과정을 f_i들의 표준편차가 0이 될 때까지 반복한다.

<표 3> Job3의 Job Data

i	nd_i	nr_i	P_i	cd_i	cr_i	k_i
1	5	3	-	3	6	1.5
2	7	2	-	4	5	1.5
3	6	4	-	2	7	0.75

<표3>의 활동 3개를 <그림 6>의 Push과정을 사용해 단축하였다. <그림 7>의 예시를 보면 Related Group {3,1}의 d^*가 Push 과정을 통해 최적 작업 시간을 찾음을 알 수 있다. 하지만 Push 과정의 반복은 탐색 시간을 증가시킨다. 이를 위해 최적 작업 시간(d^*)를 구하는 식을 도출한다.



<그림 7> Job3 = {2} {3,1}의 Push 과정

활동 a, b의 d^*를 구하기 위해서는 각 활동에 투입할 최대 가능 자원의 합($ar_a + ar_b$)을 계산해야 한다. 이를 위해 각 ar_a와 ar_b을 식 (2)에 대입하여 다음과 같이 계산하였다.

$$ar_a = nr_a + (nd_a - d_a) \times k_a$$
$$ar_b = nr_b + (nd_b - d_b) \times k_b$$
$$ar_a + ar_b = \sum_{i=1}^{2} nr_i + \sum_{i=1}^{2} nd_i k_i - (d_a k_a + d_b k_b)$$

위의 식에서 a와 b의 작업 시간인 d_a과 d_b가 같을 때가 최적 작업 시간(d^*)이기 때문에 d^*를 대입하여 정리하면 아래 식 (3)을 도출 할 수 있다.

$$\sum_{i=1}^{2} ar_i = \sum_{i=1}^{2} nr_i + \sum_{i=1}^{2} nd_i k_i - (d^* k_a + d^* k_b)$$
$$\sum ar_i = \sum nr_i + \sum nd_i k_i - d^* \sum k_i$$
$$d^* = \frac{\sum nr_i + \sum nd_i k_i - \sum ar_i}{\sum k_i} \quad (3)$$

도출된 d^* 식을 적용하여 다음 장에 새롭게 Push Algorithm을 정리한다.

(3) d^* 식을 적용한 Push Algorithm 설계

본 연구에서 제안하는 d^*가 고려된 Push Algorithm을 아래 <표4>에 pseudo code로 나타냈다. 가장 먼저 SSGS로 배치된 Sub Group의 배치 기간(SST, SFT)를 확인한다. 그 후 해당 범위에 포함된 활동들의 작업 정보를 생성하고, 배치할 활동들 간의 관계를 확인하여

<표 4> Push Algorithm pseudo code

```
Push Algorithm(PerArray[p], Sub[g],C)
1   SST←Group Completion Time_min, SFT←Group Completion Time_max
2   b←1
3   while(b < C) do
4       JobStart(JS), JobFinish(JF) ←Job Data
5       if((JS ≤ SST&&JF ≤ SST)||(SFT ≤ JS&&SST ≤ JF)) then
6       Else
7          EJ[] : ExistJob[] ←Job Data
8       end if
9          b = b+1
10  end while
11  con[] : Controljob[] ←sub[]
12  rel[] ← Create Related Group(con[])
13  e←1
14  while(e < E) do
15      Calculate d*
16      if(d* is available) then
17         Resource and Time of con[]←d*
18      Else
19         Resource and Time of con[]←Push(rel[e])
20      end if
21         e←e+1
22  end while
23  Return Result of Push Algorithm
```

Related Group을 생성한다. 기존의 <그림 6>의 Push 과정은 d^*가 적용이 불가능할 때만 Push를 실행하며, d^*가 적용이 가능하다면 적용하고 작업 정보들을 업데이트 한다.

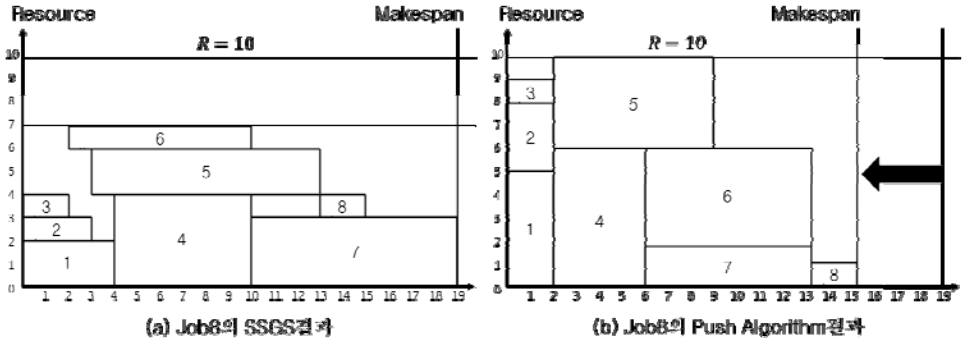
4.2.4 SSGS와 Push Algorithm의 통합

일정 계획을 생성하는 방법인 SSGS와 d^*가 고려된 Push Algorithm을 통합한 Algorithm의 절차를 아래의 <표 5>의 pseudo code로 나타내었다. 기존의 연구에서는 유한한 (시간, 자원)의 조합들을 각 활동에 할당한 후에 SSGS를 통해 일정 계획을 생성 했지만, 본 연구에서 제안된 Algorithm은 가장 적은 자원(nr_i)을 투입한 활동들을 SSGS를 통해 배치하고, 이를 Push Algorithm을 통해 최적의 (시간, 자원)을 도출하도록 한다. Jia and Seo(2013b)는 정수 기반의 RCPSP에 Facility Layout Problem의 개념을 가져와 활동들을 기존의 SSGS를 통해 일정 계획 공간(x축:시간, y축:자원)에 배치한 후 활동 간의 배치 위치의 변화를 주는 Shift Operator를 통해 각 시간(t)에 투입된 자원의 사용율을 최대로 만들어 전체 makespan을 최소화 하는 방법을 사용하였다. 본 연구에서는 Push 과정에서 배치 순서(PerArray)의 변화를 통해 각 활동들이 일정 계획의 공간에서 어떤 위치에 있어야 makespan이 최소로 단축되는지 확인한다.

<그림 2>의 Job8을 기반으로 생성된 솔루션 {1,2,3} {4,5} {6,7} {8}을 통해 제안된 Algorithm을 실행하였다. <그림 9>와 같이 SSGS를 통해 계획을 작성했을 때 19(a)였던 makespan이 15.22(b)라는 값으로 단축되었음을 확인할 수 있다. 해당 값은 <그림 9>의 (a)에서 최대한 자원을 투입하여 얻을 수 있는 최소값의 makespan 값이다.

<표 5> SSGS and Push Algorithm pseudo code

SSGS and Push Algorithm(Solution(l))	
1	Sub[] : SubGroupArray [1,2,…..,G] ←Solution(l)
2	g←1, C←1
3	while($g < G$) do
4	PerArray[1,2…,P] ←Permutation(sub[g])
5	p←1
6	while($p < P$) do
7	SSGS(PerArray[p], sub[g])
8	Push Algorithm(PerArray[p],sub[g], C)
9	BestSolutionOfSubGroup ← PushSolution
10	p←p+1
11	end while
12	c←c+jobcountOfsubgroup
13	g←g+1
14	end while



<그림 8> Job8의 Push 결과

4.3 Simulated Annealing with Heuristic Algorithm

RCPSP를 해결하기 위해 기존의 연구에서 Branch & Bound, Genetic Algorithm, Tabu Search, Simulated Annealing 등 다양한 Local Search 방식이 적용되고 있다. 그러나 DTRTP에서는 Tabu Search(De Reyck et el, 1998), Genetic Algorithm(Ranjbar, 2007), Scatter Search(Ranjbar et el, 2008) 이외에 다른 Local Search 방식이 적용되어지지 않았다. 본 연구에서 Simulated Annealing 기반 Heuristic Algorithm을 제안하여, 기존의 연구된 Genetic Algorithm과 비교한다.

Simulated Annealing은 Kirkpatrick et el(1983)이 개발한 확률적인 Heuristic Algorithm이다. 많은 경우의 수가 존재하는 조합최적화 문제 해결에 적합하다. 담금질 방식에서 초기 온도의 냉각 속도를 조절하는 방식을 차용하였다. 해당 방식은 조합최적화 문제에서 지역 최적해에 빠지지 않고 전역 최적해에 근접한 해를 얻을 수 있다.

4.3.1 기본 설정

기본적인 Simulated Annealing 방식에서는 초기 온도와 냉각 속도를 설정하고, 무작위로 탐색된 이웃 해의 값이 좋으면 해의 개선과 동시에 초기 온도를 냉각한다. 이러한 반복은 초기 온도가 한계 온도에 도달할 때까지 반복한다. 최적해의 도출을 위해서 적정한 냉각 속도 조정과 초기온도, 한계 온도 설정이 중요하다. 이를 위해 다음과 같은 실험과 방법을 적용하였다.

(1) 냉각 속도 조정 방법(Cooling Schedule)

Simulated Annealing는 초기 온도에서 한계 온도로 냉각이 되면서 해 값의 변화가 적어지며, 최적값으로 수렴하게 된다. 냉각 속도를 너무 빠르게 하면 최적값으로 너무 빠르게 수렴하여 다양한 해를 고려하지 못하게 되어 지역 최적해에 빠질 가능성을 증가시킨다. 가장 기본적인 냉각 속도 조정 방법으로 기하 스케줄 방식이 있다. 기하 스케줄 방식은 $T_{x+1} = \alpha T_x$ 으로 해 값의 개선에 따라 냉각 변수 α $(0 \le \alpha \le 1)$를 초기온도에 적용하여 냉각시킨다. 하지만 기하 스케줄은 너무 빠르게 온도가 냉각되어 많

은 해의 영역을 검색하지 못하는 경우가 발생한다. 이를 위해 본 연구에서는 적응적 기하스케줄(Ortner et el, 2007)을 통해 냉각 속도를 조정 한다. 적응적 기하 스케줄은 해의 개선율($\langle E\rangle_{x+1}$)이 이전 온도의 개선율($\langle E\rangle_{x}$)보다 높을 경우 현재 온도(αT_{x+1})를 냉각하고, 이전보다 낮거나 같은 경우($\langle E\rangle_{x+1} \leq \langle E\rangle_{x}$)에는 냉각을 멈추는 방법이다. 이 경우, 해의 개선이 안 일어날 경우, 온도를 멈춰서 높은 온도에서의 개선 가능성을 늘릴 수 있다. 하지만 해의 값이 최적값이 도달한 경우, 온도가 멈추고 Algorithm이 무한 반복하는 오류가 발생한다. 이를 위해 해의 개선이 일어나지 않는 경우, 온도 냉각율(β)을 0.999로 하여 매우 천천히 감속하도록 한다. 이는 다음과 같은 식 (4)로 표현된다.

$$T_{x+1} = \begin{cases} \beta T_{x+1}\ ,\ \text{if}\ \ \langle E\rangle_{x+1} \leq \langle E\rangle_{x} \\ \alpha T_{x+1}\ ,\ \text{if}\ \ \langle E\rangle_{x+1} > \langle E\rangle_{x} \end{cases} \quad (4)$$

적응적 기하 스케줄을 문제에 적용하기 위해서는 적절한 냉각율(α)을 확인해야 한다. 가장 일반적으로 사용되는 냉각율은 0.95~0.99 이다. 이 중 가장 적합한 냉각율을 찾기 위해서 각 냉각율 마다 100번씩 실험을 실시하여 도출된 최적값(makespan)의 평균을 비교한다. 초기온도는 10, 한계 온도는 0.1로 설정하고, 활동 15(최적해(Best makespan) : 20.851/ 일일 한도(R) : 14) 문제에 각 냉각율을 적용하여 실험을 실시한다.

<표6>의 실험 결과를 보면 0.99의 냉각율일 때 가장 좋은 값을 보인다. 해당 실험은 자원을 최대한으로 사용하는 Push Algorithm이 적용되었기 때문에 결과값 간의 차이가 크지 않다. 하지만 무한한 조합에서 충분히 다양한 조합을 비교해보고, 그 중 좋은 조합들을 찾아 비교할 수 있는 시간이 필요하다. 0.99의 냉각율에 경우 천천히 온도를 냉각시키면서 다양한 해 값을 찾아봤기 때문에 가장 좋은 값이 나왔다. 본 연구에서는 0.99의 냉각율을 사용하여 Simulated Annealing을 실시한다.

(2) 초기 온도 및 한계 온도 설정

Simulated Annealing에서 초기 온도는 너무 높을 경우 탐색 시간이 길어지는 문제가 발생되며, 너무 낮을 경우 충분한 탐색이 이루어지지 않은 문제가 발생한다. 본 연구에서는 무한한 실수 조합에서 충분한 탐색을 근사 최적해를 찾아야 하기 때문에, 적절히 높은 초기 온도 설정의 필요성이 존재한다. 근사 최적해로의 수렴은 초기 온도(T_0)의 높이에 영향을 받기 때문이다. 적절한 초기온도 설정을 위한 실험을 위해 15개(R : 14)의 활동을 고려한 문제 표본이 사용되었다. 이를 바탕으로 일반적으로 사용되는 초기 온도(T_0)인 10에서 100까지 실험을 진행하였다. 한계 온도는 0에 가까운 값인 0.1로 설정하였다. 각 실험을 100번씩 실시하여 설정된 초기 온도 중 도출된 최적값의 평균($\overline{F}$)과 최적해에 도출까지 걸리는 평균 수행 시간(Time)을 확인한다.

<표 6> 냉각율에 따른 적응적 기하스케줄 실험 결과

냉각율(α)	0.95	0.96	0.97	0.98	0.99
실험의 평균값	21.64091	21.64077	21.52202	21.46502	21.32971

<표 7> 초기 온도(T_0)에 따른 최적값의 평균과 평균 수행 시간 결과

Result \ T_0	10	20	30	40	50	60	70	80	90	100
$\overline{F}$	21.57	21.55	21.55	21.53	21.52	21.53	21.50	21.50	21.47	21.38
Time(초)	21.11	21.34	22.13	22.68	22.89	23.41	23.79	24.22	25.71	26.23

<표 7>의 실험 결과를 보면 초기온도의 증가에 따라 최적값의 평균($\overline{F}$)이 좋아지는 것을 확인할 수 있다. 그러나 초기온도가 10에서 100까지 증가할 때 평균값이 0.88%의 개선율 만을 보이는 데에 반해, 수행 시간은 21.11초에서 26.23초로 약 1.24배 증가하는 것을 알 수 있다. 초기온도와 상관없이 최적값인 20.851을 찾기 때문에 본 연구에서는 상대적으로 적은 개선 대비 높은 수행 시간이 발생하는 100이 아닌, 가장 빠른 수행시간을 가지며 충분한 개선 결과를 보여주는 10으로 초기온도를 설정한다.

4.3.2 이웃해 탐색

가장 기초적인 이웃해 탐색으로 무작위로 선택된 활동 간의 교환을 하는 Pairwise Interchange 방식이 있다. 해당 방식은 동시에 작업하는 활동들을 묶는 방식으로 Solution을 구성하는 Sub Group Solution에서 많은 보정 문제를 일으킨다.

<표 8> Pairwise Interchange VS New Interchange

Method	Pairwise Interchange	New Interchange
Basic Solution	{1,2,3} {4,5} {6,7} {8}	
Interchange	2↔7	5↔6
New Solution	{**1**,7,3}{**4**,5}{6,2}{8}	{1,2,3}{4,6}{5,7}{8}

위의 <표 8>와 같이 2와 7을 교환 한다고 했을 경우, 1,4은 7의 선행 관계이므로 동시에 오거나 뒤에 올 수 없고, 2는 5의 선행관계 이므로 뒤에 올 수 없다. 이를 feasible한 해로 보정하기 위해 필수적으로 많은 Sub Group에서 보정 과정이 발생하게 된다. 고려하는 활동과 교환하는 활동이 많아질수록 더 많은 보정 과정이 발생한다. 이는 수많은 조합을 고려해야 하는 Simulated Annealing에서 탐색 시간을 증가시키는 요소가 된다. 이를 해결하기 위해 본 연구에서는 이웃한 Sub Group간의 교환방식인 New Interchange를 제안한다. 위의 교환 결과와 같이 교환할 활동인 5의 Sub Group {4,5}과 이웃한 {6,7}의 활동 6과 교환한다. 이러한 경우, 다른 Sub Group의 보정 과정이 발생하지 않는다. 또한 이웃한 Sub Group 활동들의 선행관계 만을 고려해 교환이 일어나기 때문에, 보정작업이 발생하지 않는다.

4.3.3 Simulated Annealing 절차

본 연구에서 제시한 이웃 Sub Group간의 교환방식을 적용한 Simulated Annealing과 makespan을 최소화하는 (시간, 자원) 최적 조합 Algorithm인 Push Algorithm과 스케줄링 방식인 SSGS를 더하여 아래 <표 9>의 pseudo code로 표현하였다.

<표 9> SA with Heuristic Algorithm pseudo code

Simulated Annealing with Heuristic Algorithm	
1	Initialization : generate a sub group solution $X \leftarrow$ CreateSubGroupSolution(L)
2	Evaluate the objective value for the current solution: $f(X) \leftarrow$ SSGS and Push Algorithm
3	$T \leftarrow T_{\max}$
4	while($T > Stop_T$) do
5	$l \leftarrow 1$
6	while($l < L$) do
7	$X' \leftarrow$ FindNeighborSolution(X)
8	$f(X') \leftarrow$ SSGS and Push Algorithm
9	$\triangle \leftarrow (f(X) - f(X'))$
10	if($\triangle < 0$) then//개선이 안된
11	$Boltsmann\,Value \leftarrow \exp(\frac{-\triangle}{T})$
12	if($Boltzmann\,Value \le Random(0,1)$) then
13	Accept the new Solution : $X \leftarrow X'$
14	end if
15	else
16	if($\langle E \rangle_{x+1} > \langle E \rangle_x$)
17	$T \leftarrow T \times \alpha$
18	else
19	$T \leftarrow T \times \beta$
20	end if
21	Accept the new Solution : $X \leftarrow X'$
22	end if
23	Best Solution Check(X')
24	$l \leftarrow l + 1$
25	end While
26	end while

Ⅴ. 실 험

기존의 연구들에서는 알고리즘의 성능을 평가하기 위해 기존의 제공된 문제 라이브러리를 바탕으로 최적해와 오차율을 비교하거나 도출된 근사 최적해의 값을 비교하였다. 본 연구의 경우 실수 기반의 자원-시간 트레이드오프 관련 기존 연구와 문제 라이브러리가 존재하지 않기 때문에 본 연구를 위해 새로운 문제들을 생성하였다. 또한 실수 값을 고려한 문제들의 특성상 무한한 조합을 고려해야하기 때문에 최적해를 찾기 어렵다. 이 때문에 본 연구에서는 도출된 근사 최적해를 비교하는 실험을 진행한다. 근사 최적해를 찾기 때문에 기존의 연구와 같이 최적해와 오차율을 비교하기 어렵다. 본 연구에서는 제안된 Algorithm의 성능을 증명하기 위해서 Ranjbar(2007)이 제안한 Genetic Algorithm과 본 연구에서 제안된 Algorithm에

서 도출된 근사 최적해(makespan)들을 비교하였다.

또한 본 연구에서 제안된 Heuristic Algorithm인 Sub Group Solution을 통한 Push Algorithm의 성능을 증명하기 위해서 Genetic Algorithm에 Sub Group Solution 구조와 Push Algorithm을 추가하여 Heuristic Algorithm의 유무에 따른 성능을 비교하였다. 이에 대한 pseudo code를 아래 <표 10>에 나타내었다.

<표 10> GA with Heuristic Algorithm pseudo code

	Genetic Algorithm with Heuristic Algorithm
1	Initialization : generate a sub group solution $X \leftarrow$ CreateSubGroupSolution(L)
2	iter←1
3	while(iter < $\text{iter}_{\max}$) do
4	Evaluate the objective value for the current solution: $f(X) \leftarrow$ SSGS and Push Algorithm
5	Roulette Wheel Selection
6	Two Point crossover
7	Insertion Mutation
8	iter←iter+1
9	end While

제안된 Algorithm(Simulated Annealing with Heuristic Algorithm)과 비교할 Algorithm(Genetic Algorithm with Heuristic Algorithm, Genetic Algorithm)은 C#으로 구현되었으며, Intel 2Ghz, 4G Ram을 가진 PC에서 실험하였다. 실험 데이터는 다양한 활동에서의 성능 비교를 위해 15, 21, 28, 35, 65개의 활동들을 선 후행 제약을 가지는 활동 데이터를 생성하였으며, 생성된 데이터 <표 11>에 각 활동의 작업시간, 투입 자원 등의 속성들은 랜덤으로 생성하였다. 각 활동마다 동일한 일일 한도 자원(Daily Limit)인 12, 15, 18이 주어졌을 때, 각 Algorithm의 결과 값을 비교한다.

<표 11> 실험 표본

Number of Activities (Size)	15, 21, 28, 35, 65
일일 자원 한도 (Daily Limit)	12, 15, 18

SA의 실험을 위해 실험 변수는 시작온도=10, 냉각속도=0.99, 임계온도=0.1 로 앞 장의 실험을 통해 설정하였다. Genetic Algorithm의 파라메터의 경우 Ranjbar(2007)의 연구를 참조하였다. 각 Algorithm의 실험은 10번 반복하였으며, 결과 값은 Best Solution의 평균값($\overline{F}$)과 Best Solution(F_{Best}), 표준편차(s), 평균 수행시간($\overline{T}$, 초)으로 표시하였다. 추가로 $\overline{F}$의 개선율을 통해 각 Algorithm 간의 개선 정도를 측정하였다. 실험결과를 아래에 일일 자원 한도(Limit)가 12<표 12>, 15<표 13>, 18<표 14>인 경우에 따라 정리하였다.

결과값의 개선정도를 확인하기 위해 실험을 통해 도출된 결과를 기존 알고리즘 대비 제안된 알고리즘의 평균값($\overline{F}$) 개선율<그림9>과 Limit별 각 알고리즘의 평균 수행시간($\overline{T}$)<그림 10>을 그래프로 나타내었다. 평균값($\overline{F}$)의 개선율 결과인 <그림 9>의 (a)와 (b)를 보면 Limit에 상관없이, 활동의 개수가 증가 할수록 기존의 연구에 비해 본 연구에서 제안한 알고리즘(ⓒ)의 개선율이 점차 증가함을 알 수 있다. 하지만 <그림 9>의 (b)를 보면 기존 GA(ⓐ)와 다르게 Heuristic Algorithm의 Push Algorithm이 적용된 GA(ⓑ)와는 개선 정도가 미미함을 알 수 있

다. 이를 통해 본 연구에서 제안한 SA with Heuristic Algorithm(ⓒ)은 기존의 정수 기반의 알고리즘인 GA 대비 실수 기반의 DTRTP에서 더 좋은 성능을 보임을 알 수 있다.

<표 12> Limit 12일 경우 실험 결과

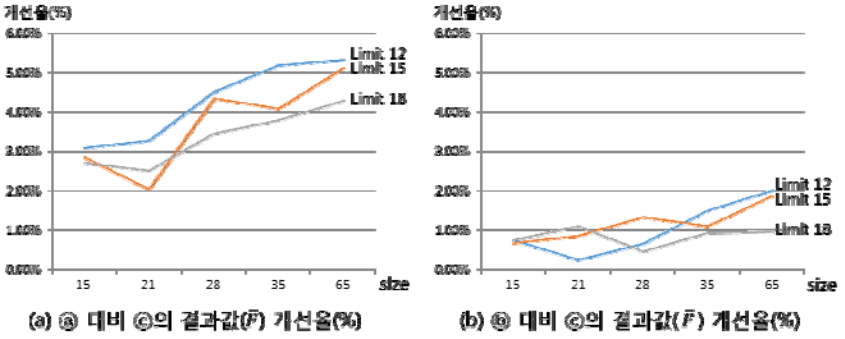
Method	(a) GA				(b) GA with Heuristic Algorithm				(c) SA with Heuristic Algorithm				(c)-(a) 개선율 (%)	(c)-(b) 개선율 (%)
size	$\overline{F}$	F_{Best}	s	$\overline{T}$(초)	$\overline{F}$	F_{Best}	s	$\overline{T}$(초)	$\overline{F}$	F_{Best}	s	$\overline{T}$(초)		
15	23.42	23.00	0.44	108	22.87	22.50	0.35	77	22.70	22.50	0.24	51	3.08%	0.75%
21	34.12	33.00	1.01	296	33.10	33.00	0.76	204	33.02	32.34	0.52	123	3.27%	0.24%
28	42.17	42.00	1.30	384	40.54	42.00	0.64	305	40.28	40.00	0.34	145	4.52%	0.65%
35	92.42	88.28	2.73	845	89.17	87.12	3.30	561	87.86	86.23	1.27	259	5.19%	1.49%
65	181.61	178.50	1.72	2784	175.88	169.25	0.99	2210	172.44	165.62	4.06	798	5.32%	1.99%

<표 13> Limit 15일 경우 실험 결과

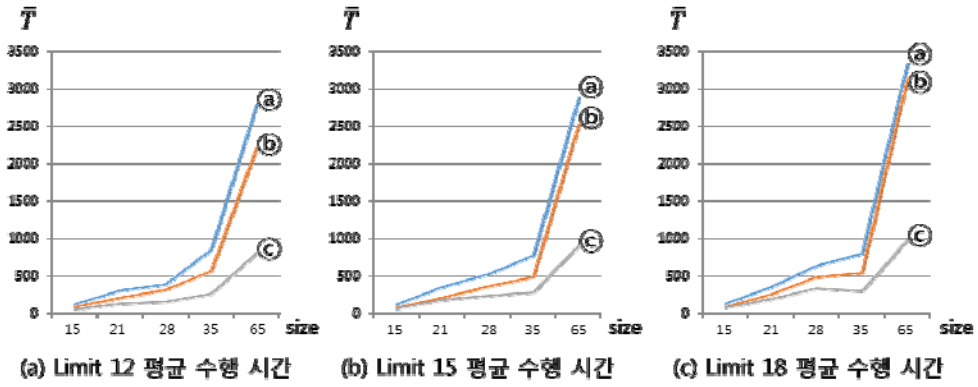
Method	(a) GA				(b) GA with Heuristic Algorithm				(c) SA with Heuristic Algorithm				(c)-(a) 개선율 (%)	(c)-(b) 개선율 (%)
size	$\overline{F}$	F_{Best}	s	$\overline{T}$(초)	$\overline{F}$	F_{Best}	s	$\overline{T}$(초)	$\overline{F}$	F_{Best}	s	$\overline{T}$(초)		
15	21.71	19.88	1.98	115	21.24	19.62	2.09	70	21.10	19.62	1.39	63	2.84%	0.66%
21	32.15	30.92	2.01	342	31.78	28.50	2.45	206	31.51	28.16	1.75	174	2.03%	0.86%
28	37.90	34.68	3.84	510	36.80	34.00	3.27	354	36.32	32.55	3.65	232	4.35%	1.32%
35	82.11	73.00	8.05	764	79.75	72.00	6.94	478	78.89	68.82	8.17	271	4.08%	1.09%
65	165.53	146.23	15.72	2870	160.38	137.29	16.55	2516	157.46	135.60	15.32	894	5.11%	1.85%

<표 14> Limit 18일 경우 실험 결과

Method	ⓐ GA				ⓑ GA with Heuristic Algorithm				ⓒ SA with Heuristic Algorithm				ⓒ-ⓐ 개선율 (%)	ⓒ-ⓑ 개선율 (%)
size	$\overline{F}$	F_{Best}	s	$\overline{T}$(초)	$\overline{F}$	F_{Best}	s	$\overline{T}$(초)	$\overline{F}$	F_{Best}	s	$\overline{T}$(초)		
15	20.88	19.13	1.46	101	20.49	18.13	2.52	79	20.33	18.13	1.67	69	2.71%	0.74%
21	29.86	28.00	3.32	344	29.45	26.50	3.28	248	29.13	26.36	2.63	184	2.51%	1.10%
28	34.20	33.78	5.36	632	33.20	29.33	5.32	475	33.06	28.00	4.54	227	3.45%	0.45%
35	75.32	63.90	10.47	792	73.22	61.46	10.23	528	72.55	59.40	10.53	289	3.79%	0.92%
65	153.68	132.47	19.83	3326	148.78	122.43	21.73	3132	147.37	121.73	21.26	972	4.28%	0.96%



<그림 9> GA(ⓐ)와 GA with Heuristic Algortihm(ⓑ) 대비
SA with Heuristic Algorithm(ⓒ)의 평균값 개선율



<그림 10> Limit별 알고리즘의 평균 수행 시간

<그림 10>을 보면, 본 연구에서 제안된 알고리즘(ⓒ)이 기존 GA(ⓐ)와 Sub Group Solution과 Push Algorithm이 적용된 GA(ⓑ) 대비 빠른 수행시간을 보여줌을 알 수 있다. 특히 활동 개수가 15개일 경우 ⓐ 대비 최대 1.7배, ⓑ 대비 최대 1.5배의 평균 수행 속도를 보여주지만, 활동 개수가 65개로 늘어날 경우 ⓐ 대비 최대 3.4배, ⓑ 대비 최대 3.2배까지 빠른 수행 속도를 보여줌을 알 수 있다. 수행 속도의 차이는 Solution 구조와 이웃해 탐색 방식에서 발생하는 보정작업으로 인해 발생한다. 기존의 GA에 경우 이웃해 탐색을 위한 crossover시 부모해를 통해 자식해를 생성하게 되는데 이때, 선행 제약을 만족하는 가능해를 생성하기 위해 보정작업이 발생한다. 이는 활동 개수가 늘어날수록 더 많은 보정 작업이 발생하게 되며, 수행 시간의 증가로 나타나게 된다. 이러한 보정 작업의 발생을 방지하기 위해서 crossover 대신 정성욱과 김준우(2016)가 후보순위 방식을 이용한 해 생성 방식을 제안했고, 김준우(2016)는 복호화 방식을 적용하였으나, DTRTP의 연구에서는 적용된 적이 없다. 본 연구에서 제안된 SA의 경

우 이웃해 탐색 시 이웃한 Sub Group 간의 교환이 가능한 활동만을 고려하기 때문에 보정작업이 발생하지 않는다. 이로 인해 기존 GA 대비 더 빠른 수행속도를 보여준다.

위의 실험 결과를 통해 본 연구에서 제안한 SA with Heuristic Algorithm의 Push Algorithm은 활동의 개수가 증가할수록 기존의 GA 대비 실수 차원의 DTRTP에서 더 좋은 근사 최적해를 도출하며, Sub Group Solution구조와 SA의 이웃 Sub Group Interchange를 통한 이웃해 탐색 방식은 기존 GA 대비 더 빠른 수행 시간 내의 결과를 도출한다.

Ⅵ. 결 론

본 연구에서는 정수 기반의 유한한 (시간, 자원)의 조합을 다루는 기존의 DTRTP 연구와 다르게 실수 값을 고려한 (시간, 자원)의 무한한 조합에서 makespan을 최소화하는 조합을 도출하는 Simulated Annealing 기반 Heuristic Algorithm을 제안했다. 실수 차원에서 최적 조합을 도출하기 위해서 기존의 방식처럼 유한한 조합들을 활동에 할당하는 것이 아니라 각 활동의 위치 비교를 통해 (시간, 자원)의 최적 조합을 도출하는 Push Algorithm을 개발했다. 또한 활동들의 배치순서만을 고려하는 기존의 Permutation Solution 구조와 다르게 동시에 배치되는 활동들을 고려하는 Sub Group Solution 구조와 이를 통해 이웃해를 탐색하는 이웃 Sub Group Interchange를 Simulated Annealing에 적용했다. 제안된 Algorithm은 기존의 Genetic Algorithm 대비 고려하는 활동의 개수가 많아질수록 높은 개선율(%)을 보여주며, 상대적으로 빠른 수행시간을 가지는 것으로 분석됐다.

본 연구는 기존의 연구되지 않았던 실수 기반의 시간-자원의 관계를 다루고 있어, 실수 값의 시간이나 자원을 고려해야 하는 실시간 일정 계획 문제나 현실에 가까운 자원 배치 계획에서 적용 될 수 있을 것이다. 그러나 본 연구의 Algorithm은 기존의 연구가 존재하지 않아 하나의 자원만을 고려했다는 한계점이 존재한다. 이러한 점은 실제로 다양한 자원들이 고려되는 프로젝트 스케줄링 문제를 해결하는데 한계가 있다. 추후 연구에서는 다중 자원을 고려한 실수 기반의 DTRTP Algorithm 개발이 연구 주제가 될 수 있다.

참고문헌

김준우. “Job Shop 일정계획 문제 풀이를 위한 유전 알고리즘의 복호화 방법”. 정보시스템연구, 25(4), 2016, pp105-119.

정성욱, & 김준우. “후보순위 기반 타부 서치를 이용한 제약 조건을 갖는 작업 순서결정 문제 풀이” 정보시스템연구, 25(1), 2016, pp159-182.

Bell, C. E., & Park, K., “Solving resource – constrained project scheduling problems by a* search.” Naval Research Logistics (NRL), 37(1), 1990, pp. 61-84.

Blazewicz, J., Lenstra, J. K., & Kan, A. R., “Scheduling subject to resource constraints: classification and complexity.” Discrete applied

mathematics, 5(1), 1983, pp. 11-24.

Christofides, N., Alvarez-Valdés, R., & Tamarit, J. M., “Project scheduling with resource constraints: A branch and bound approach.” European Journal of Operational Research, 29(3), . 1987. Pp. 262-273.

Ciesielski, V., & Scerri, P., “Real time genetic scheduling of aircraft landing times.” In 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360), 1998, May, pp. 360-364

De Reyck, B., Demeulemeester, E., & Herroelen, W., “Local search methods for the discrete time/resource trade-off problem in project networks.” Naval Research Logistics (NRL), 45(6), 1998, pp. 553-578.

Demeulemeester, E., & Herroelen, W., “The discrete time/resource trade-off problem in project networks: a branch-and-bound approach”. IIE transactions, 32(11), 2000, pp. 1059-1069.

Dewi, D. S., & Septiana, T., “Workforce scheduling considering physical and mental workload: a case study of domestic freight forwarding.” Procedia Manufacturing, 4, 2015, pp. 445-453.

Hong, K. S., & Leung, J. T., “On-line scheduling of real-time tasks.” In Proceedings. Real-Time Systems Symposium, IEEE, 1988, December, pp. 244-250.

Icmeli, O., & Erenguc, S. S., “A tabu search procedure for the resource constrained project scheduling problem with discounted cash flows.” Computers & operations research, 21(8), 1994, pp. 841-853.

Ortner, M., Descombes, X., & Zerubia, J. “An adaptive simulated annealing cooling schedule for object detection in images”. 2007, Research Report, INRIA, Vol. 6336

Jia, Q., & Seo, Y., “An improved particle swarm optimization for the resource-constrained project scheduling problem.” The International Journal of Advanced Manufacturing Technology, 67(9-12), 2013a, pp. 2627-2638.

Jia, Q., & Seo, Y., “Solving resource-constrained project scheduling problems: conceptual validation of FLP formulation and efficient permutation-based ABC computation.” Computers & Operations Research, 40(8), 2013b, pp. 2037-2050.

Kelley, J. E., “The critical-path method: resource planning and scheduling.” Industrial scheduling. 1963.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P., “Optimization by simulated annealing.” science, 220(4598), 1983, pp. 671-680.

Kolisch, R., Sprecher, A., & Drexl, A.

"Characterization and generation of a general class of resource-constrained project scheduling problems." Management science, 41(10), 1995, pp. 1693-1703.

Kolisch, R., & Hartmann, S., "Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis." In Project scheduling, Springer, Boston, MA, 1999, pp. 147-178.

Ranjbar, M. R., & Kianfar, F., "Solving the discrete time/resource trade-off problem in project scheduling with genetic algorithms." Applied Mathematics and Computation, 191(2), 2007, pp. 451-456.

Ranjbar, M., De Reyck, B., & Kianfar, F., "A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling." European Journal of Operational Research, 193(1), 2009, pp. 35-48.

Sprecher, A., Kolisch, R., & Drexl, "A. Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem." European Journal of Operational Research, 80(1), 1995, pp. 94-102.

김 건 아 (Kim, Geon-A)

성결대학교 산업경영공학 학사를 취득하였다. 현재 고려대학교 산업경영공학과 석사과정으로 재학 중이며, 주요 관심분야는 Algorithm, 생산시스템, 프로젝트 스케줄링 등이다.

서 윤 호 (Seo, Yoon-Ho)

고려대학교 산업공학학사와 펜실베니아 대학교 산업공학석사와 산업공학박사학위를 취득하였다. 현재 고려대학교 산업경영공학과 교수로 재직하고 있으며, 주요 관심분야는 제조시스템 공학, CAPP 등이다.

<Abstract>

Development of a Heuristic Algorithm Based on Simulated Annealing for Time-Resource Tradeoffs in Project Scheduling Problems

Kim, Geon-A · Seo, Yoon-Ho

Purpose

This study develops a heuristic algorithm to solve the time-resource tradeoffs in project scheduling problems with a real basis.

Design/methodology/approach

Resource constrained project scheduling problem with time-resource tradeoff is well-known as one of the NP-hard problems. Previous researchers have proposed heuristic that minimize Makespan of project scheduling by deriving optimal combinations from finite combinations of time and resource. We studied to solve project scheduling problems by deriving optimal values from infinite combinations.

Findings

We developed heuristic algorithm named Push Algorithm that derives optimal combinations from infinite combinations of time and resources. Developed heuristic algorithm based on simulated annealing shows better improved results than genetic algorithm and further research suggestion was discussed as a project scheduling problem with multiple resources of real numbers.

Keyword: time-resource tradeoffs, project scheduling, simulated annealing, real numbers

* 이 논문은 2019년 11월 6일 접수, 2019년 11월 17일 1차 심사, 2019년 12월 2일 게재 확정되었습니다.