Journal of the Korean Institute
of Industrial Engineers
Vol. 18, No.2, Dec., 1992

작업 스케쥴링 문제 해결을 위한 Branch & Bound 해법의 비교분석

고석주*, 이채영*

Heuristic Aspects of the Branch and Bound Procedure for a Job Scheduling Problem

Seok Joo Koh* & Chae Y. Lee*

Abstract

This article evaluates the efficiency of three branch-and-bound heuristics for a job scheduling problem that minimizes the sum of absolute deviations of completion times from a common due date. To improve the performance of the branch-and-bound procedure, Algorithm SA* is presented for the initial feasible schedule and three heuristics : breadth-first, depth-first and best-first search are investigated depending on the candidate selection procedure. For the three heuristics the CPU time, memory space, and the number of nodes generated are computed and tested with nine small examples ($6 \leq n \leq 4$). Medium sized random problems($10 \leq n \leq 30$) are also generated and examined. The computational results are compared and discussed for the three heuristics.

1. 서 론

최근 작업 스케쥴링의 문제[2, 3, 4]에서는 due date 이후의 작업뿐 아니라 due date 이전의 작업에 대해서도 penalty를 부과하는 스케쥴링 문제를 많이 다루고 있다. penalty는 각 작업별 due date와 작업완료시간과의 차이의 제곱값이나 절대값으로 계산되는데 본 논문에서는 후자를 다루기로 한다.

주어진 due date로부터 작업 완료시각과의 절대편차합을 최소화하는 작업 스케쥴링 문제는 제한된 경우(restricted dase)와 제한되지 않은 경우(unrestricted case)로 분류된다[2]. 각 작업 i에 대하여

P_i : 작업 처리시간($i=1, 2, \cdots, n$)

d : 납기일(due date)

Ci : i작업의 완료시각

$MS = \sum_{i=1, \cdots, n} P_i$

$f(S) = \sum_{i=1, \cdots, n} | C_i - d |$

* 한국과학기술원 경영과학과

라 할때 문제는 위의 목적함수 f(S)를 최소화하는 sequence를 찾는 것이다.

P_i를 $P_1 \le \cdots \le P_n$으로 나열할때

$$\triangle = \begin{cases} P_1+P_3+\cdots+P_n : n\text{이 홀수} \\ P_2+P_4+\cdots+P_n : n\text{이 짝수} \end{cases}$$

에서, $d \ge \triangle$이면 제한되지 않은 경우로써 V형 스케쥴에 의한 최적 알고리즘[2]이 알려져 있고 O(n logn)의 계산으로 구해진다. $(P_1+P_2)/2 < d < \triangle$이면 제한된 경우로써 NP-Complete이고 enumeration 이외에는 일반적인 해법이 알려져 있지 않다. 여기서 $d \le (P_1+P_2)/2$의 경우는 P_1과 P_2 두 작업 중 짧은 작업을 먼저 처리하는 쪽이 페널티가 적으며 나머지 작업들도 작업 처리시간이 짧은 순서로 배치하였을 때 총 페널티가 최소화 된다[3]. 따라서, SPT(shortest processing time)에 의하여 최적해가 구해진다. Enumeration에 의한 해법은 작업수가 많아질수록 처리시간과 기억용량이 기하학적으로 증가하는 단점을 가지고 있어 그 효율성이 떨어진다. 따라서 가능한 한 계산비용을 줄일 수 있는 휴리스틱이 사용되는데 이 논문에서는 제한된 경우에 대하여 알려져 있는 Branch and Bound(B & B) 알고리즘을 구현함에 있어 따르는 계산시간과 기억공간(memory space) 문제를 비교, 분석 하고자 한다.

2. 문제의 성격 및 접근방법

2.1 기존 알고리즘의 소개

B & B에 의한 알고리즘을 소개하기 전에 제한된 작업 스케쥴링 문제에 대하여 기존의 연구[2, 3]에서 밝혀진 몇가지 중요한 특징을 살펴보면 다음과 같다. 여기서 첫번째 작업의 시작시각은 '0'으로 가정한다.

1. 최적 sequence는 V형태를 따른다. 즉, 최적 sequence를 Q라 할때 가장 짧은 job을 중심으로 $Q=\sigma 1 \pi$라 놓으면 σ는 LPT로 나열되고 π는 SPT로 나열되며 작업간 여유시간(idle time)은 없다.

위의 성질은 검색 과정에서 due date 이전의 LPT(Longest processing time)부분만 결정하면 뒤의 SPT(Shortest Processing Time) 부분은 자동으로 결정되게 하며 탐색범위를 감소시켜 준다.

2. $Q=\sigma i 1 j \pi$ (i=2 또는 j=2)가 최적 sequence이면 due date는 $C(\sigma)+(P_1+P_2)/2 \le d \le C(\sigma_1)+(P_1+P_1)/2$를 만족한다. 이때, $C(\sigma)$는 σ에 속하는 모든 작업을 마쳤을 때의 완료시각이다.

위의 성질을 다시 표현하면,

$T_1 = C(\sigma_1)-(P_1-P_1)/2 \le C(\sigma_1)+(P_1+P_1)/2 = T_2$

에서

$d-(P_1+P_1)/2 \le C(\sigma_1)d+(P_1-P_1)/2$

즉, 작업 1은 $(d-(P_1+P_1)/2,\ d+(P_1-P_1)/2)$에서 시작해야 한다.

3. 최적 sequence Q를 $\sigma a \pi$라 할때 (σ : LPT, π : SPT이고 1, 2, 3을 포함하지 않는다) σ 다음, 또는 π앞에 올 수 있는 작업 i는 $i \ge u$를 만족해야 한다. 이때 u는

$H_u(\sigma) = \sum_{k \in \sigma} P_k + (P_1+P_u)/2 \sum_{k=2}^{u-1} P_k$라 할때,

$H_u(\sigma) < d \le H_{u+1}(\sigma)$를 만족하는 가장 작은 작업이다.

4. $\sigma i (i>3)$로 시작하는 sequence를 고려하자. $(i, i+1, i+2, \cdots, n)$에서 σi를 제외한 작업들을 SPT로 나열한 sequence를 β라 할때

$\sum_{k \in \beta} P_k \ge MS-d$를 만족하면, σi에 대한 V형 Sequence는 $\sigma i \gamma \beta$ 하나밖에 없다. ($\gamma = i-1, i-2, \cdots, 1$)

위에서 소개된 제한된 경우의 작업 스케쥴링 문제의 특성을 이용하여 Szwarc[3]은 B & B 알고리즘을 제시하였다. 본 연구에서는 위 B & B 알고리즘의 적용에서 가능한 다음의 세가지 휴리스틱 측면 [5]을 고려한다.

첫째, level 단위로 기억과 후보선택이 이루어지는 breadth-first와 둘째, 목표에 도달하거나 목표를 찾지 못하고 한 후보문제의 완전한 해를 구할때까지 가지를 치는 depth-first, 그리고 최적해에 빨리 근접할 수 있는 방향으로 즉, 최소화 문제의 경우 lower bound(LB)가 작은 후보를 우선으로 후보선택이 이루어지는 best-first가 있다.

위 네가지를 특성을 고려한 일반적인 B & B 알고리즘에 depth-first heuristic을 적용한 흐름도(flow chart)는 [그림 1]과 같다. [그림 1]의 흐름도중 ⓐ~ⓕ에 해당하는 부분에 대한 알고리즘의 구현과정은 다음과 같다.

ⓐ SA(Sundararaghavan & Ahmed) 알고리즘에 의한 f(S)를 초기 upper bounnd UBO로 하고 σ(SPT에 해당하는 sequence)를 공집합으로 한다. 여기서 SA알고리즘에 의한 해법은 다음과 같다. 구간 (a, b)에서 배치되야 할 작업을 고려할 때 L=d-a, R=b-d를 각각 due date를 기준으로 왼쪽, 오른쪽 구간 길이라 하자. 여기서 R≤L이면 남은 작업 중 가장 긴 작업을 맨 오른쪽에 L>R이면 맨 왼쪽에 배치한다. 초기값으로는 a=0, b=MS를 주고 모든 작업이 배치될 때까지 바꾸어 준다.

ⓑ upper bound를 구하고 끝낸(fathomed) 마디에 대하여 UBO와 크기를 비교하여 더 작으면 바꾸어 준다.

ⓒ 일차원 배열 $LB(\sigma i)=\sum_{k\in\sigma i}|C_k-d|+\sum_{k\in\beta}|C_k-d|+f_{i-1}$

여기서 $f_1=f_{i-2}+\sum_{i=1,\ i-1}P_k$, $f_1=0$, $f_2=P_1$

ⓓ 일차원 배열 A1을 만들어 들어온 순서대로 sequence를 기억한다.

ⓔ i는 u에서 s까지 변화되고 이때 s+1은 σ의 마지막 원소이다.

즉, σ가 공집합일 경우 s는 n이 된다.

ⓕ 기억된(저장된) 마디에 대하여 다음 작업을 위한 후보선택 과정으로 세 가지 휴리스틱에 따라 다르다.

2.2 SA알고리즘의 개선

작업 수가 많아질수록 SA 알고리즘에 의한 해는 최적해에 가까와진다. SA에 의한 sequence를 S, 최적 Sequence를 Q라 하면 상대편차 (f(S)−f(Q))/f(Q)는 0에 근사한다. 이에 기초하여 SA sequence에서 V형을 유지하며 bound를 개선시킬 수 있는 방법(SA*)을 제시한다.

Algorithm SA*

Step1. SA에 의한 sequence $S=\sigma 1\pi$를 구하며 upper bound의 값 UBO를 계산한다.

Step2. i←n

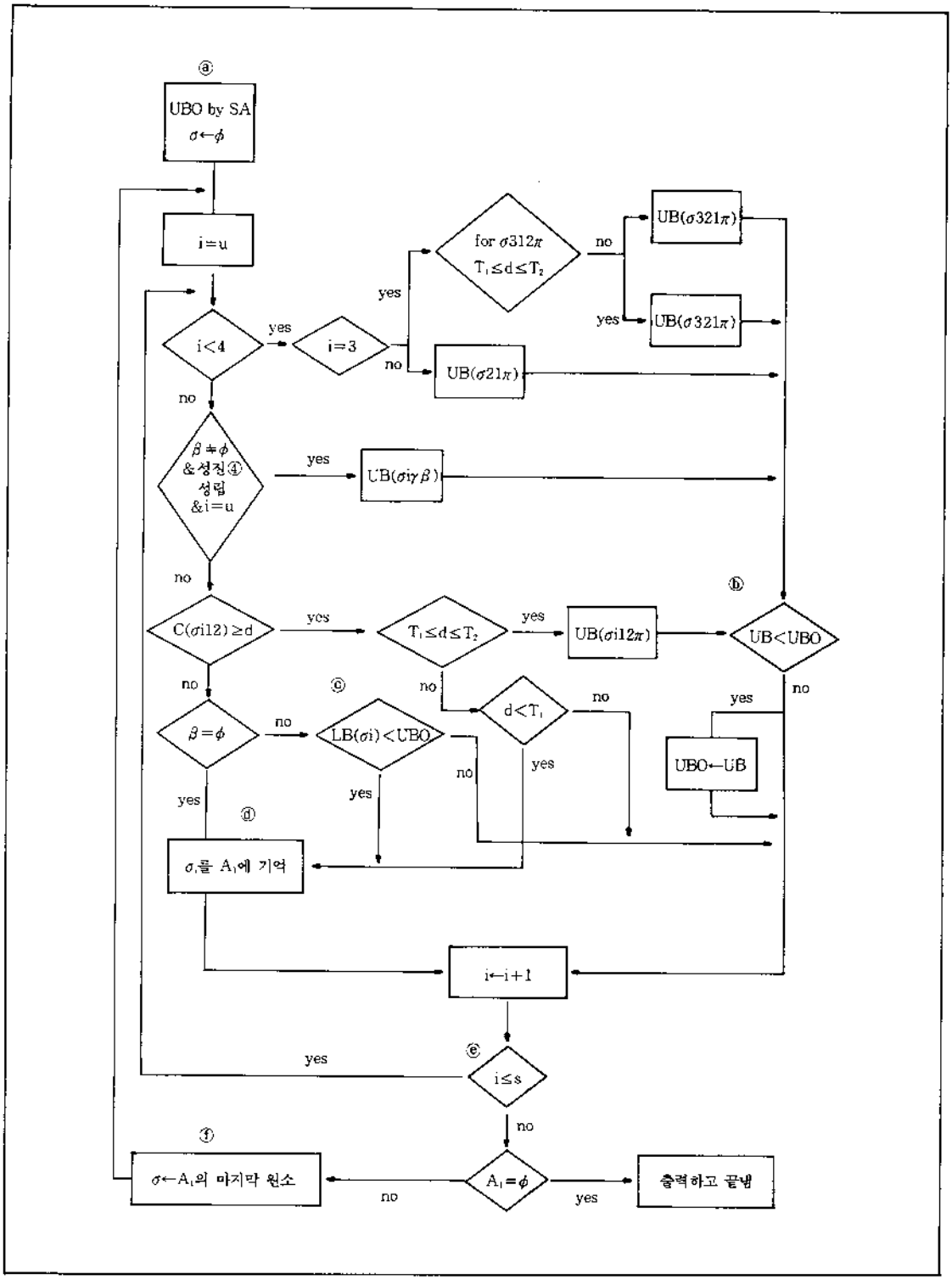
Step3. job i와 job i-1이 같은 집합 σ 또는 π에 있으면 Step6으로 간다.

Step4. 그렇지 않으면 i와 i-1의 위치를 교환하여 S′이라하고, upper bound UB를 계산한다.

Step5. UB<UBO 이면 UBO←UB, S←S′

Step6. i←i−1, i=1이면 끝내고, 그렇지 않으면 Step3으로 간다.

위의 알고리즘에서 job i를 고려할 때 i와 i−1이 같은 집합에 있는 경우는 상호 교환하였을 때 (i−1, i)가 되어 V형을 위반한다. 따라서 서로 다른 집합에 있을 경우에 한하여 상호 교환하고 그때의 upper bound의 값을 비교하는 것이다. 이는 순서가 인접한 두개의 작업에 대해서만 기존의 upper bound와 비교 개선하며 i가 1이 될 때까지 계속된다.



[그림 1] Depth-first에 따른 B & B알고리즘

2.3 세가지 휴리스틱의 비교

흐름도에서 후보선택이 이루어지는 과정을 제외한 나머지 부분은 세 가지 휴리스틱에 대하여 모두 동일하게 적용된다. 세가지 휴리스틱의 차이를 다음의 예제[4]를 중심으로 살펴보면 다음과 같다.

예제) n=9, P_i=3, 6, 7, 8, 14, 30, 45, 48, 72, i=1, …, 9, d=130

1) Breadth-first

레벨 단위의 저장을 위하여 일차원 배열 A_1, A_2를 만든다. σ가 공집합일때 u가 7이 되어 7, 8, 9의 작업에 대하여 검색이 이루어지며 가지를 치는 마디 9를 A_1에 저장한다. 다음 레벨에서 A_1의 9를 A_2로 옮기고 σ=9에 대하여 검색한다. 이때 가지를 치는 97, 98을 역시 A_1에 기억하고 다음 레벨에서 A_2로 옮긴다. 레벨3에서는 σ=97, 98에 대하여 검색을 한 결과 974만 가지를 친다. 레벨 4에서는 더 이상 가지를 치는 마디가 없으므로 끝낸다. 검색 도중 upper bound 값이 UBO 보다 작으면 UBO로 바꾼다.

레벨 1 : A2={ }, A1={9}

레벨 2 : A2={9}, A1={97, 98}

레벨 3 : A2={97, 98}, A1={974}

레벨 4 : A2={974}, A1={ }→A2={ }

2) Depth-first

레벨에 상관 없이 A_1만 있고 레벨 2에서 레벨 3으로 갈때 나중에 저장된 98을 먼저 검색한다. 이는 기억용량의 효율면에서 의미가 있다. 97보다는 98로 시작되는 마디, 즉 작업시간이 큰 쪽을 먼저 검색할수록 깊이(depth)가 짧아 현행 기억용량이 줄어들게 된다. 만약 i가 u에서 시작하지 않고 s에서 시작한다면 나중에 저장된 u가 먼저 검색되므로 기억용량을 더 차지할 것이다. 98에서 가지를 치지 않으므로 다음에 97을 검색한다. 이러한 방법으로 A_1이 공집합일 때까지 계속한다.

Step1 : {9}→9선택

Step2 : {97, 98}→98선택

Step3 : {97}→97선택

Step4 : {974}→974선택

Step5 : { }

3) Best-first

저장된 후보 중에서 lower bound(LB)가 가장 작은 후보를 택한다. 이 방법에 의하면 LB자체를 계산하는 데에 시간이 많이 걸리며 레벨이 깊어질수록 LB가 작아진다는 보장이 없어(LB(σi)< LB(σij)일 수도 있음) 계산시간의 효율면에서 불리하다.

Step1 : {9}→하나밖에 없으므로 9선택

Step2 : {97, 98}→LB(97)=231>LB(98)=169, 98선택

Step3 : {97}→97선택

Step4 : {974}→974선택

Step5 : { }

3. Computation 결과

세가지 휴리스틱에 대한 우위는 상대적이고 문제의 성격마다 다르지만 처리시간과 기억용량 면에서는 depth-first가, 검색되는 총 마디수면에선 best-first가 유리하다고 알려져 있다.

본 연구에서는 세가지 휴리스틱의 효율성을 비교하기 위해 [4]에서 주어진 9개의 예제와 randomly generated된 문제들에 대하여 처리시간(CPU time), 동시에 저장되는 A1원소의 최대 갯수(Maximum # of nodes stored), 그리고 검사된 총 마디수(total # of nodes generated)를 조사하였다. 또한 SA와 SA*의 performance를 분석하였다.

모든 프로그램의 실행은 CONVEX C1-XP에서 이루어 졌으며 9개 예제의 결과는 〈표 1〉과 같다.

〈표 1〉 Small Example[4]에 대한 세 휴리스틱의 비교

	Upper Bound			CPU time in seconds			max. # of nodes stored			total # of nodes generated		
n	d/△	SA*	SA	Breadth	Depth	Best	Breadth	depth	Best	Breadth	depth	Best
6	189	198	189	0.067	0.000	0.000	2	2	2	8	8	8
6	355	360	360	0.033	0.000	0.000	2	2	2	8	8	8
6	387	397	393	0.067	0.000	0.000	2	2	2	10	10	10
6	265	307	307	0.033	0.000	0.000	2	2	2	6	6	6
9	274	275	274	0.083	0.000	0.000	2	2	2	18	18	18
14	1092	1094	1094	0.250	0.117	0.117	16	8	15	175	175	175
14	1603	1606	1606	0.400	0.217	0.250	52	13	31	448	409	409
14	1742	1746	1742	0.500	0.333	0.417	70	13	42	646	631	633
14	1820	1821	1821	0.317	0.183	0.200	36	10	24	364	364	364

〈표 2〉는 n=10, 20, 30에 대한 결과이다. 작업처리시간 Pi는 1에서 100사이의 uniform 분포를 따르게 하였으며 d/△=0.3, 0.6, 0.9의 경우에 대하여 각각 10회 실행한 결과의 평균치이다.

이상 30개까지의 계산결과를 종합하면, 총 검색된 마디수 면에서는 알고리즘의 특성상 모두 비슷하고, 처리시간 면에서는 best-first 방법을 제외한 나머지 방법의 효율이 비슷한 것으로 나타났다. 기억용량 면에서는 depth-first 방법이 매우 효율적이다. 또한 SA* 알고리즘에 의한 초기해는 SA에 의한 해와 비교할때 문제의 크기가 커질수록 최적해에 가까운 것으로 나타나 그 효율성이 SA에 비해 우월한 것을 판단된다.

〈표 2〉 Random Example에 대한 세 휴리스틱의 비교

10^6(f(S)−f(Q)/f(Q)				CPU time in seconds			max. # of nodes generated			total # of nodes generated		
n	d/△	SA*	SA	Breadth	Depth	Best	Breadth	depth	Best	Breadth	depth	Best
	0.3	868	980	0.080	0.018	0.015	3	3	3	17	17	17
10	0.6	706	1683	0.014	0.030	0.025	8	5	7	52	52	52
	0.9	3572	5100	0.143	0.030	0.035	11	6	8	68	65	67
	0.3	18	197	0.513	0.398	0.447	51	13	41	681	678	679
20	0.6	26	132	3,540	3,509	4,706	456	20	321	5543	5529	5543
	0.9	376	491	7,663	7,553	12,573	1121	26	628	11448	11218	11414
	0.3	26	105	14.47	14.85	22.52	975	31	869	19749	19694	19729
30	0.6	11	125	389.8	382.5	3134.3	24131	46	22961	503905	497520	501860
	0.9	13	142	1889	1709	191650	143870	62	133092	2269569	2095179	2107015

f(S) : objective function value of initial scheduling by SA or SA*

f(Q) : optimal value by the branch & bound

참 고 문 헌

[1] Baker, R. 1974. Introduction to Sequencing and Scheduling, *Wiley*.

[2] Baker, R., and D. Scudder. 1989. Sequencing With Earliness And Tardiness Penalties; Review, *Operation Research*. 38. 22~35.

[3] Szwarc, W. 1989. Single-Machine Scheduling to Minimize Absolute Deviation of Completion Times from Common Due Date. *Naval Res. Logist. Quart. 34*, 663-673.

[4] Bagchi, U., Y. Chang and R. Sullivan. 1987. Minimizing Absolute Deviations of Completion Times about a Common Due Date. *Naval Res. Logist. Quart. 34*, 277-240.

[5] Parker G. R., and R, L, Rardin. 1988. Discret Optimization. *Academic Press*.