공학석사 학위 논문

적치장 내 블록 반입·반출일정 기반 최적 블록 적치 시스템

Optimal Block Stacking System Based on the Schedule of Block Insertion and Removal in the Shipyard

울 산 대 학 교 대 학 원

전기전자컴퓨터공학과

김 근 완

적치장 내 블록 반입·반출일정 기반 최적 블록 적치 시스템

지 도 교 수 권 영 근

이 논문을 공학석사 학위 논문으로 제출함

2023년 7월

울 산 대 학 교 대 학 원

전기전자컴퓨터공학과

김 근 완

김근완의 공학석사 학위 논문을 인준함

심사위원장 조 동 식 (인)
심사위원 김 종 면 (인)
심사위원 권 영 근 (인)

울 산 대 학 교 대 학 원

2023년 7월

[국문 요약]

조선소에서 블록을 보관하는 적치장 환경은 한정적인 환경에서 제한적인 선택이 뒤따른다. 언제나 적치장 공간이 부족 현상이 발생하고 이에 대한 블록 이동이 필연적으로 발생한다. 그래서 조선소에서 블록의 보관 그리고 불필요한 이동을 감소시키는 것은 항상 조선소에서 고민하는 분야이다. 이를 해결하기 위해 적치장 내 블록 반입 및 반출 일정을 고려한 블록 적치가 상당히 중요하다.

이와 관련된 기존 연구에서 적치장을 지번 기반으로 표현하여 블록 적치 문제에 대해 접근하였다. 이러한 관련 연구들에 대한 한계점으로는 블록의 크기 대비 공간의 활용이 떨어졌다는 점이다. 또한 반입·반출 일정을 적용은 배제된 채 진행된 관련 연구들은 반출 시 발생하는 간섭 블록이 최적화되지 않았다는 한계점도 뚜렷하게 도출해 낸다. 이러한 기존 지번 기반의 문제해결 방식을 개선하고자 이 논문에서 제시한 연구는 좌표 기반 그래프로 표현하여, 주어진 블록 반출·반입 일정에 따라 블록 적치를 최적화하는 방법을 제안하였다. 따라서 공간 효율성의 최대화와 간섭 블록의 수를 최소화하는 휴리스틱 알고리즘을 제안하였다. 본 논문에서 적치장 블록 적치 문제에 대한 핵심 아이디어는 블록 별로 하나의 정점으로 표현되는 그래프를 새로 생성한 후, 탐색알고리즘을 적용한다. 모든 이동 가능 정점을 얻는다. 그리고 출입구까지 거리를 정의하여 일정표를 통한 블록의 초기 반입 위치를 도출한다. 또한 시뮬레이션을 통해 제안하는 방법이 다른 알고리즘에 공간 효율성과 간섭 블록의 수가 나은 결과를 나타낸다.

주제어: 조선소 적치장, 블록 적치 최적화, 간섭 블록 최소화, 블록 반입, 블록 반출

목차

표 및 그림 목차

Ⅰ. 표 목차

Ⅱ. 그림 목차

1. 서론

1.1 연구 목적 및 필요성

조선소에서 선박을 건조 하는 과정에 있어서 소규모 블록부터 대규모 블록에 이르기까지 조립, 의장, 도장, 탑재 과정을 거쳐 선박을 완성한다. 블록들은 다양한 모양을 가지며 이러한 블록들을 보관하는 공간을 적치장이라고 한다. 이 적치장은 블록만을 적치하는 것만 아니라 다른 부자재 같은 것들도 보관하기도 하며, 선박 건조 계획에 따라 사용되는 시점까지 보관된다.

조선소는 하나의 선박을 건조 하는 것이 아니라 여러 선박을 건조하기 때문에 상황에 따라 많은 블록이 발생한다. 하지만 조선소에서 적치장은 한정적으로 주어지기 때문에 공간이 부족하면 도로의 공간을 활용한 적치가 빈번하게 발생하기도 한다. 이와 같은 환경적인 한계점과 현실적인 요인들을 감안한 조선소 내에 선박건조과정의 조건들을 생각하면 적치장 공간에 대한 효율적인 블록 적치는 중요한 고민거리이다. 또한 매번 생산되는 블록들을 일정에 맞춰 적치장에서 최적의 위치에 보관하기가 힘들다. 그래서 블록들은 작업이 필요한 시점에 필요한 장소로 운반되기 위해 반출되는 과정에서 간섭 블록이 발생한다. 간섭 블록이란 필요한 블록을 적치장에서 반출하기 위해서 어쩔 수 없이 임시로 반출해야 하는 비효율적인 블록이다. 간섭 블록을 반출하기 위해서는 추가로 트랜스포터가 배정되어 작업이 이뤄지므로 비용 측면에서 손해가 발생한다. 그래서 간섭 블록 발생을 최소화하도록 적치장을 일정에 맞게 효율적으로 운영하는 것이 중요하며, 그러기 위해서 블록 운반 관리와 적치장 공간 활용이 중요하다.

블록 운반 관리는 블록의 반입과 반출 일정에 따른 블록 적치 위치 결정 및 블록 반출 경로 설정 등을 말한다. 예를 들어 늦게 반입된 블록이 금방 반출될 블록을 막게 되면 다시 늦게 들어온 블록을 이동시키고 반출해야 하는 간섭 블록이 발생하기 때문에 시간과 비용이 더 들어가게 된다. 적치장 공간 활용은 말 그대로 공간 활용을 뜻한다. 조선소 내에

서 적치장은 일반적으로 지번 기반으로 구성되어 있다. 지번을 이용함으로써 관리가 용이하다는 장점이 있어, 블록이 한 적치장에 어떤 위치에 존재한다는 것을 쉽게 파악해 낼 수 있다. 반면 공간의 효율성이 떨어지고 블록을 이동하는데 한계가 있다.

본 논문은 좌표 기반 적치장을 통해 블록을 적치함으로써 좌표 단위 조절을 통한 블록의 섬세한 이동과 적치장 공간 효율성을 높이도록 하였다. 또한 좌표 기반 표현을 통해 반입할 수 있는 모든 좌표와 최적의 경로를 구한다. 그리고 반입 시 출입구의 위치와 반입 순서를 고려한 반입 알고리즘과 반출 시 발생하는 간섭 블록의 수를 최소화하는 반출 알고리즘을 제안한다.

1.2 논문의 구성

본 논문의 구성은 다음과 같다. 2장에서 기존 연구에 대하여 기술한다. 3장에서는 본 논문의 제안 방법에 대하여 기술한다. 4장에서는 실험을 통해 제안된 알고리즘의 성능을 평가한다. 5장에서는 본 논문의 결론과 향후 연구에 관해 기술한다.

2. 관련연구

조선소 내의 효율적인 블록 적치장 운영을 위한 여러 연구가 진행되었다. 예를 들면, 블록의 다양한 크기 및 이동 방법을 고려하여 블록 운반 문제를 정의하고, 적치장을 좌표로 표현하여 반출 계획 선정에 타부서치(tabu search) 알고리즘을 적용하였다[1]. 하지만 블록이 수직, 수평 이동을 하기 위해서 다른 블록이나 벽면에 붙어야 한다는 제약 조건으로 인해 실제로는 최적의 경로라고 볼 수 없다. 또한 반출 작업에 대해서만 다루고 있으며, 반입 작업에 대해서는 다루고 있지 않다는 한계점이 있다. 적치장 내에서 발생하는 블록들의 재배치 비용을 최소화하기 위해 유전알고리즘 기반 적치장의 최적 배치 방법이 제안되기도 하였다[2]. 해당 연구는 반입과 반출 작업을 모두 적용하였지만, 지번 기반으로 위치를 표현함으로써 이동이 단순하고 실제 문제에 적용하는데 제약 조건이 많다. 적치 가용 면적의 최대화와 간섭 블록의 최소화를 목표로 하는, 또 다른 유전 알고리즘을 방법이 제안되었다[3]. 해당 연구는 반입 위치를 선정할 때 적치 가용 면적을 이용하였다. 하지만 스케줄 규모가 커지면 작업 순서를 고려하여 반입한 게 아니기 때문에 상황에 따라 간섭 블록의 수가 많이 증가한다는 한계점이 있다. 동적으로 변화하는 상황을 감안하여 근사해를 도출하는 휴리스틱(heuristic) 알고리즘도 제안되었으나[4] 해당 논문 역시 지번을 이용하였고, 한 열로만 반입, 반출이 가능하다는 점에서 실제 문제에 적용하기 힘들다.

3. 제안하는 반입·반출 최적화 방법

3.1 문제표현

본 연구에서는 주어진 일정표(schedule)에 따라 적절한 반입 위치 선정을 통해 적치장 공간의 효율성 최대화와 반출 시 간섭 블록의 수를 최소화하고자 한다. 간섭 블록의 수는 블록이 반출되기 위해 반드시 간섭되는 블록의 수이다. 표 1은 최적화를 위해 초기에 입력으로 주어지는 일정표의 예시이다. 그 표에서 보듯이 각 작업(task)은 작업 종류(type), 블록 번호(id), 블록 가로 및 세로 크기(w,h) 그리고 적치 위치(x,y), 작업 시간(time)으로 구성되어 있다. 이때 작업 종류가 IN은 반입 작업을, OUT은 반출 작업을 의미하고, 일정표는 블록의 작업 시간(time) 순서로 정렬된다. 초기 일정표에서 반출 작업에 (x,y)값이 유효한 블록은 해당 위치에 이미 적치되어 있음을 뜻하고, 무효한 블록은 최적화를 통해 위치가 결정되어야 함을 뜻한다.

no.	type	id	time	size (w,h)	location (x,y)
1	IN	5	2023-04-12 06:05	(4, 7)	NA
2	IN	3	2023-04-12 06:25	(5, 4)	NA
3	IN	6	2023-04-12 06:40	(6, 3)	NA
4	IN	8	2023-04-12 07:00	(2, 5)	NA
5	IN	10	2023-04-12 07:00	(3, 4)	NA
6	OUT	12	2023-04-12 07:10	(5, 7)	(25, 12)
7	IN	16	2023-04-12 07:15	(4, 4)	NA
8	OUT	5	2023-04-12 07:20	(4, 7)	NA
9	OUT	2	2023-04-12 07:20	(5, 5)	(13, 6)
10	IN	7	2023-04-12 07:35	(7, 3)	NA
⋮	⋮	⋮	⋮	⋮	⋮

표 1. 최적화 전 주어진 초기 일정표

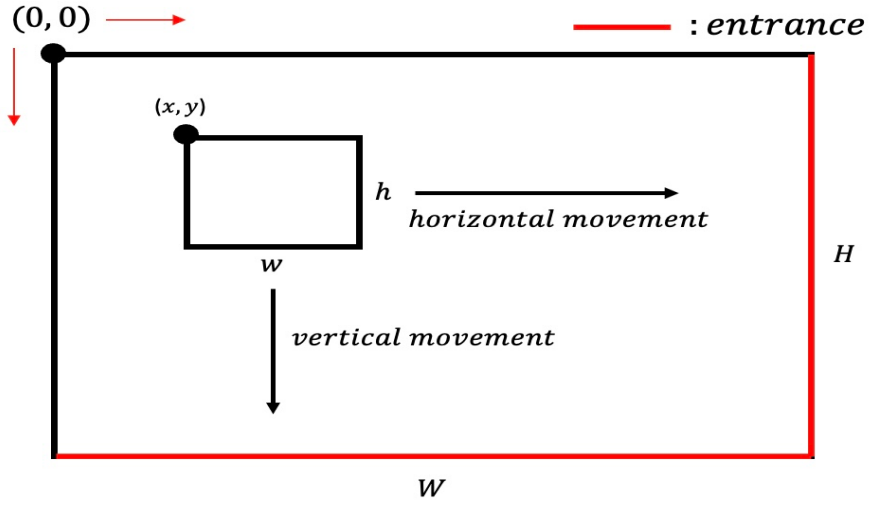

그림 1. 블록과 좌표 기반 표현

본 연구에서는 그림 1과 같이 공간 활용도와 이동성을 높이기 위해 지번이 아닌 좌표 기반 블록 위치 표현 방법을 사용한다. 적치장은 직사각형 모양으로 가정하며, 검은색 변은 출입 불가, 붉은색 변은 출입구임을 뜻한다. 적치장 좌표는 왼쪽 상단을 원점으로 한다. 높이가 H, 폭이 W인 적치장의 블록 점유 상태를 표현하기 위해 2차원 이진배열인 $map[0 \ldots W-1][0 \ldots H-1]$을 정의한다($H$와 W은 정수). 즉, $map[i][j] = 0$은 (i, j)을 왼쪽 상단, $(i+1, j+1)$을 오른쪽 하단으로 하는 기본 크기 격자 공간이 비어 있음을 뜻하고, $map[i][j]=1$은 반대로 블록이 점유함을 뜻한다. 만약 가로 크기는 h, 세로 크기는 w인 어떤 블록의 위치가 (x,y) (단, x와 y는 음이 아닌 정수)일 때, $map[i][j]=1$ $(x \le i \le x+w-1, y \le j \le y+h-1)$이 된다. 그림 2는 적치장에서 2차원으로 표현된 블록들을 2차원 이진배열인 map으로 표현한 예를 보여준다.

블록의 이동 방향은 수평 이동과 수직 이동이 가능하다. 본 논문에서는 사선 이동을 고려하지 않는데 그 이유는 좌표 단위를 작게 조정하면 수평 수직 이동을 통해 사선 이동과 비슷한 효과를 낼 수 있기 때문이다. 이후에 적치 위치 및 반출경로 탐색 알고리즘을 적용하기 위해 블록

이동 가능 그래프 $G(V,E)$를 정의한다. 이때 $V=\{(x,y)|\,0\le x\le W-w, 0\le y\le H-h, x$와 y는 정수$\}$이고 $E\subseteq\{((x_1,y_1),(x_2,y_2))|(x_2=x_1+1,\ y_2=y_1)$ 또는 $(y_2=y_1+1,\ x_2=x_1)\}$이다. 즉 $((x_1,y_1),(x_2,y_2))\in E$이면 해당 블록이 (x_1,y_1)과 (x_2,y_2)사이에 이동할 수 있음을 의미한다. 그림 3은 크기가 $w=2, h=1$인 블록에 대해 블록 이동 그래프의 예를 보여준다.

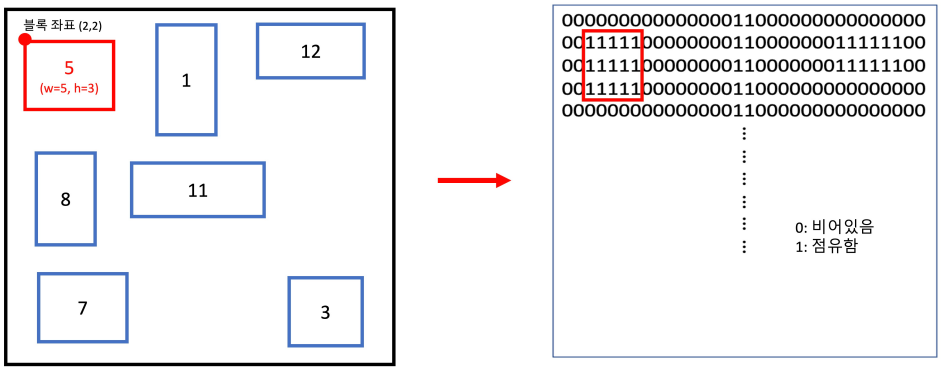

그림 2. 적치장 상태 표현

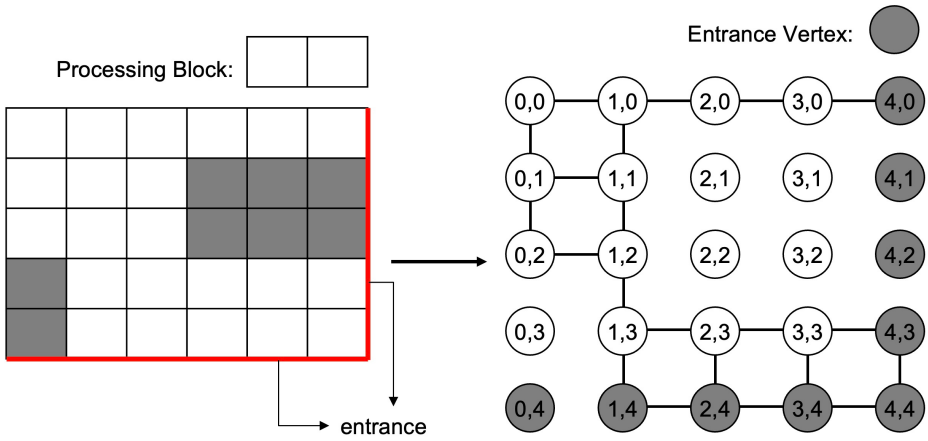

그림 3. 블록 이동 그래프

3.2 최적화 문제 정의

그림 4는 블록 적치 최적화 알고리즘의 전체적인 과정을 설명한다. 블록 반입·반출 최적화를 적용하기 위해 일정표(schedule)와 적치장 점유 상태(map), 적치 블록 정보(stock)가 입력으로 사용된다. 이때 stock은 적치장에 적치된 블록들 번호(id) 집합이다. R은 반입 블록들의 초기 출입구까지 거리 집합이다. C_I는 반입 작업 시 반입 불가 블록의 수를, C_O는 반출 작업 시 발생하는 간섭 블록의 수를 의미한다. 일정표에서 일정 하나씩 시간 순서로 진행이 되며, 해당 일정의 작업 종류에 따라 반입 및 반출 알고리즘이 각각 적용된다. 반입 일정일 경우 종료 후 반입 과정에서 발생한 반입 불가 블록의 수가 누적된다. 반면 반출 일정일 경우 종료 후 반출 과정에서 발생한 간섭 블록의 수가 누적된다. 따라서 본 논문에 다루는 문제는 C_I와 C_O을 최소화하도록 반입 블록의 위치를 결정하는 것이다.

```
OptimizeLocation(schedule, map, stock){
 C_I, C_O ← 0
 Initialize R from schedule, map and stock
 for i = 1 to T:
   if schedule[i].type = IN: // 반입 일정일 경우
     C_I ← C_I + insert(schedule, i, map, stock, R)
   if schedule[i].type = OUT: // 반출 일정일 경우
     C_O ← C_O + remove(schedule, i, map, stock)
 return C_I, C_O
}
```

그림 4. 메인 루틴 의사코드

3.3 반입 알고리즘

블록 반입 시 적치 위치를 최적화하는 것이 블록 반출 시 발생하는 간섭 블록의 수에 큰 영향을 끼친다. 일정을 고려하지 않고 블록의 적치 위치를 정하면, 이후 반입되는 블록으로 인해 반출 시 간섭 블록의 수가 증가하기 때문이다. 따라서 일정을 고려한 블록 반입 위치 선정이 필요하다.

출입구에서 가까운 블록들은 반출 시 간섭 블록이 적게 발생하기 쉬운 반면, 먼 블록들은 반출 시 간섭 블록이 많이 발생하기 쉽다. 따라서 먼저 반출이 될 블록을 출입구에 가깝게 적치하면 간섭 블록의 수를 줄일 수 있다. 이를 구현하기 위해 먼저 적치 블록이 전혀 없는 적치장에서 단위 크기 블록(즉, 1×1크기)에 대한 이동 가능 그래프G(3.1절 참조)를 도출한다. 각 격자 좌표의 출입구로부터의 거리를 G에서 해당 좌표 정점으로부터 출입구 정점 중 하나에 이르는 최단 경로 길이로 정의한다. 그림 5는 $W=5, H=5$인 적치장에 대해 단위 크기 이동 가능 그래프 예를 보여준다.

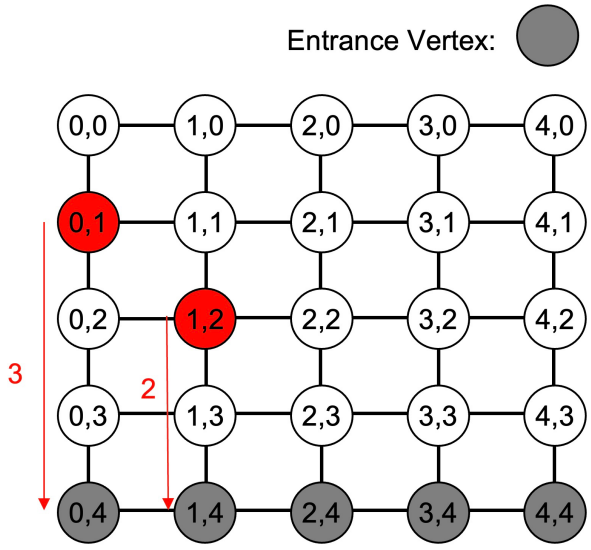

그림 5. 단위 크기 이동 가능 그래프

그림 6은 출입구 방향이 다른 적치장의 격자 거리를 시각화한 예이다(0부터 1까지 정규화하여 보인 예시이다). 출입구는 사각형의 상하좌우 모든 방향이 될 수 있으며, 출입구로부터 가까운 곳은 거릿값이 작고, 먼 곳은 거릿값이 크다. 모든 방향이 출입구인 경우 중앙부분이 가장 거릿값이 크다.

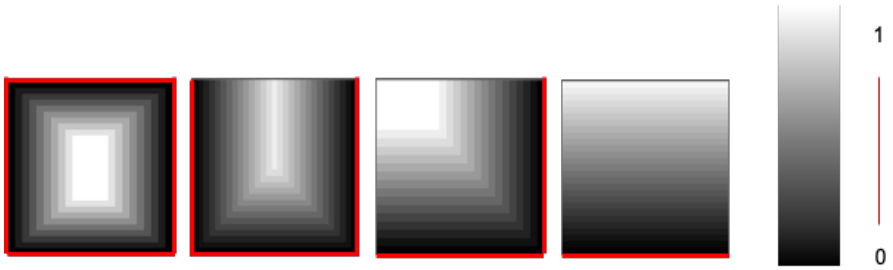

그림 6. 출입구 방향 다른 적치장의 격자 거리 시각화

이러한 격자좌표의 출입구까지 거리를 이용하여 크기가 $w \times h$인 블록(B)이 (x,y)위치에서 출입구까지 거리($\triangle$)를 다음과 같이 정의할 수 있다.

$$\triangle(B,(x,y)) = \frac{\sum_{i=0}^{w-1}\sum_{j=0}^{h-1} d(x+i, y+j)}{w \times h}$$

이때 $d(x, y)$은 좌표(x, y)의 출입구까지 거리를 뜻한다.

그림 7은 일정표에 따른 블록의 반입·반출순서 쌍에 관한 산점도이다. x축은 일정표에서 반입 순서이며, y축은 반출 순서이다. 이미 적치되어 있고, 반출 계획이 없는 블록은 반출 순서가 최댓값에 위치한다.

B는 일정을 진행하면서 현재 반입되는 블록이라 정의한다. B를 기준

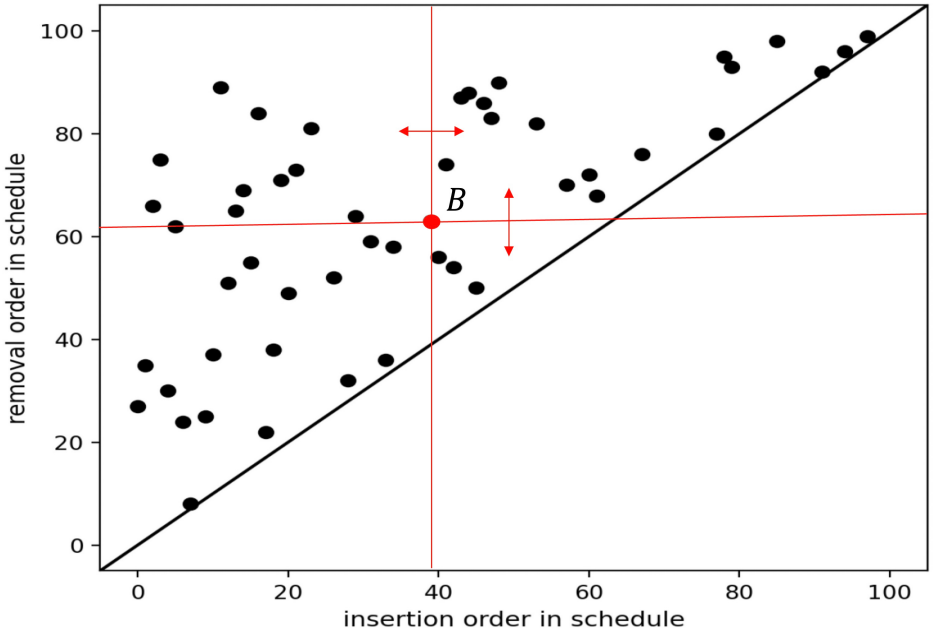

그림 7. 일정표에 따른 블록의 반입 · 반출순서 쌍에 관한 산점도

으로 수평선을 그렸을 때, 밑에 해당하는 영역은 B보다 일찍 반출되는 블록 집합이고, 위에 해당하는 영역은 B보다 늦게 반출되는 블록 집합이다. 그래서 B는 일찍 반출되는 블록들보다 출입구에서 멀게, 늦게 반출되는 블록들보다 출입구에서 가깝게 반입하는 것이 유리하다. 이때 B를 기준으로 수직선을 그렸을 때, 왼쪽에 해당하는 영역은 반입 되어있어 출입구까지 거리를 구할 수 있지만, 오른쪽에 해당하는 영역은 반입되어있지 않아 늦게 반출되는 블록들과 출입구까지 거리를 비교할 수가 없다. 그래서 각 반입 블록에 초기 출입구까지 거리를 주어 비교하였다. 기 적치된 블록은 출입구까지 거리가 있기 때문에 제외한다. 각 반입 블록별 초기 출입구까지 거리는 일정표에서 반입되는 블록 중 가로와 세로가 가장 큰 블록(B_{max}) 기준으로 빈 적치장에서 최소 출입구까지 거리($\min\triangle(B_{max})$)와 최대 출입구까지 거리($\max\triangle(B_{max})$)를 도출한다. 그리고 일정표에서 반입 블록의 수만큼 나누고, 반출 순서에 따라 곱하여 구한

다. 그러면 반출 순서가 빠른 순으로 블록별 초기 출입구까지 거리 집합(R)을 도출한다. 그림 8은 초기 출입구까지 집합 도출의 예를 보여준다.

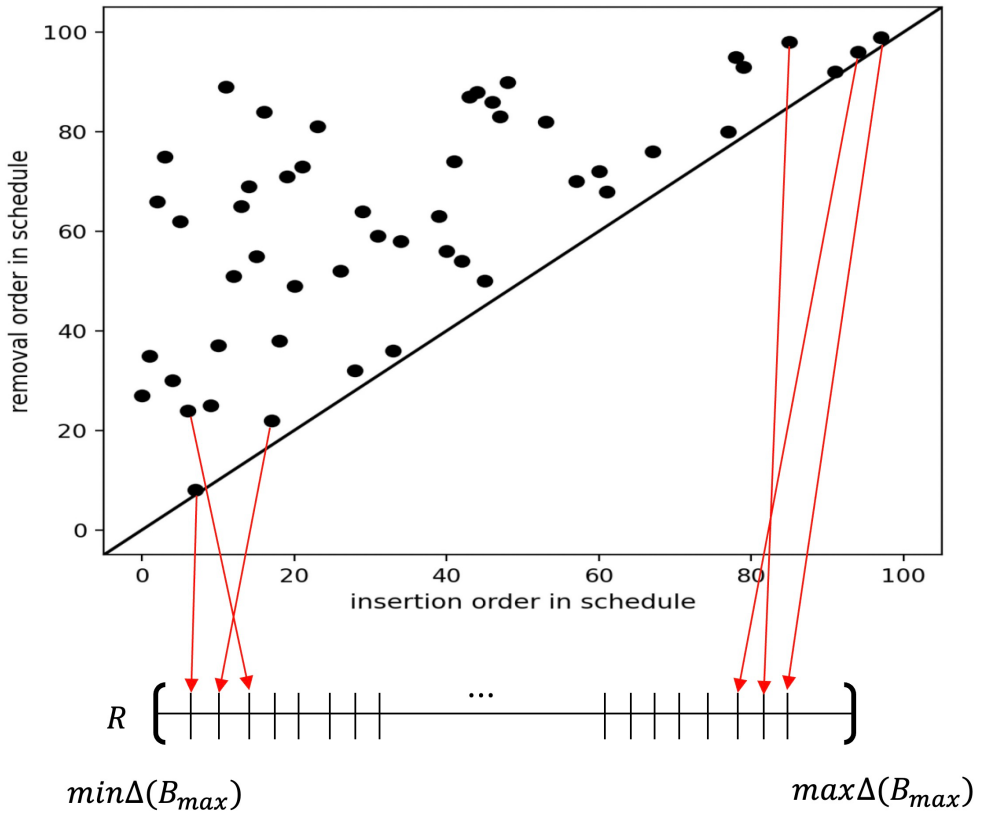

그림 8. 초기 출입구까지 거리 집합

R을 통해 B의 출입구까지 최적 거리 집합(S)과 대안 집합(A)을 도출한다. R에서 R_B기준으로 왼쪽으로 한 칸씩 이동하면서 R_B보다 작은 값이 있으면 최소 초기 출입구까지 거리($\min\Delta(B)$)로 정의하고, 오른쪽으로 한 칸씩 이동하면서 큰 값이 있으면 최대 초기 출입구까지 거리($\max\Delta(B)$)로 정의한다. 만약 $\min\Delta(B)$이 없는 경우 $\max\Delta(B)$보다 작은 블록의 출입구까지 거리 집합이 S가 되고, 나머지가 A가 된다.

max△(B)이 없는 경우 min△(B)보다 큰 블록의 출입구까지 거리들이 S가 되고, 나머지가 A가 된다. min△(B)와 max△(B) 둘 다 있는 경우 둘 사이에 해당하는 블록의 출입구까지 거리들이 S가 되고, 나머지가 A가 된다. 그림 9는 min△(B)와 max△(B) 가 둘 다 있는 경우에 S와 A 구성의 예를 보여준다.

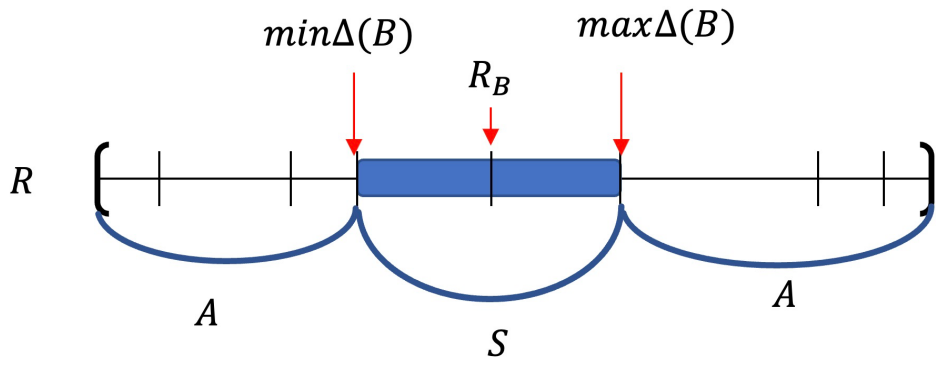

그림 9. 최소 초기 출입구까지 거리와 최대 초기 출입구까지 거리 둘 다 있는 경우의 거리 집합

반입 블록의 위치가 결정되면 적치장 상태에 따라 최적의 위치에만 적치된 게 아니기 때문에 R의 갱신이 필요하다. R_B을 기준으로 왼쪽 집합 원소들 중 △(B,(x,y))보다 크다면 △(B,(x,y))로 변경하고, 오른쪽 집합 원소들 중 △(B,(x,y))보다 작다면 △(B,(x,y))로 변경하고 R_B을 △(B,(x,y))으로 변경한다.

블록이 해당 좌표에 적치될 때 출입구와 이어진 통로를 고려하지 않으면, 다음에 반입될 블록 좌표의 가능성을 방해할 뿐만 아니라, 공간의 효율성을 낮추게 된다. 따라서 출입구와 이어진 통로의 확보가 필요하다. 이를 위해 적치되기 전의 $G(B_{\max})$와 적치 후 $G(B_{\max})$을 비교했을 때, 적치 후 $G(B_{\max})$의 노드가 개별적으로 끊어진 게 없어야 한다. 그러면 블록이 가득 차기 전에 출입구와의 통로는 계속 확보가 된다. 그림 10은 출입구와의 통로가 확보 되지 않는 예를 보여준다.

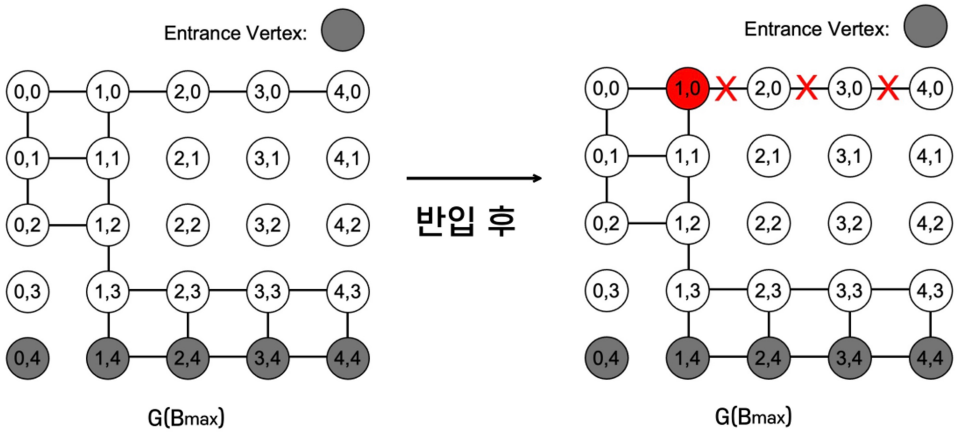

그림 10. 출입구와의 통로가 확보 되지 않는 예

그림 11은 이러한 반입 알고리즘 과정을 설명한다. 블록 이동 가능 그래프를 통해 해당 블록이 반입될 수 있는 모든 좌표의 집합(P)을 찾는다. $P=\varnothing$인 경우 반입 불가 블록이며 1을 반환한다. $P\neq\varnothing$인 경우 P에 따른 출입구까지 거리의 집합을 구한다. 그리고 S와 A에 해당하는 $P(S)$와 $P(A)$를 구한다. 먼저 $P(S)$에서 통로를 확보하는 조건을 만족하는 좌표들이 있다면, 그중 출입구까지 거리가 가장 큰 좌표를 반입 위치로 결정한다. 만약 $P(S)$에서 통로를 확보하는 조건을 만족하는 좌표들이 없다면 $P(A)$에서 통로를 확보하는 조건을 만족하는 좌표들이 있다면 그 중 R_B와 가장 가까운 좌표를 반입 위치로 결정한다. 그래도 없다면 반입 불가 블록이며 1을 반환한다. 블록의 반입 위치가 결정되면 R을 $\triangle(B,(x,y)$으로 갱신한다. 적치장 점유 상태에 블록을 추가하고, 해당하는 블록의 일정에 적치 좌표를 갱신하며 0을 반환한다.

```
insert(schedule, i, map, stock, R){

B ← schedule[i]
Construct the block movability graph G(V,E) from B and map
P = {(x, y)| ∃a path from (x,y) to an entrance vertex in G(V,E)}

if P = ∅ return 1
else
  if P(S) ≠ ∅
     (x, y) ← argmax_{(x, y)|P(S)} Δ(B, (x, y))
  else
     (x, y) ← R_B ≈_{(x, y)|P(A)} Δ(B, (x, y))
  Update to R from Δ(B, (x, y))
  map[x : x+B.w, y : y+B.h] = 1
  stock ← stock ∪ B.id
  B.location ← (x, y)
  return 0

}
```

그림 11. 반입 알고리즘 의사코드

3.4 반출 알고리즘

반출 시 간섭 블록의 개수를 최소화하기 위해서는 단순히 출입구까지의 최단 거리 반출 경로를 구해서는 안 된다. 그림 12는 간섭 블록의 개수를 최소로 하는 반출 알고리즘 과정을 설명한다. 텅 빈 맵을 가정한 블록 이동 가능 그래프로부터 구한 모든 가능한 반출 경로 중 간섭 블록의 개수가 최소인 경로(π)를 탐색한다. 각각의 간섭 블록에 대해 적치장에서 반출 후 다시 반입을 통해 적치 위치를 최적화하는 작업을 추가한다. 경로 π에서 발생하는 간섭 블록의 개수만큼 비용을 증가시킨다.

```
removal(schedule, i, map, stock){

 B ← schedule[i]
 Construct the block movability graph G(V,E) where
    V = {(x,y)|0 ≤ x ≤ W−1, 0 ≤ y ≤ H−1} and
    E ={((x1, y1), (x2, y2))|(x2 = x1 +1, y2 = y1) or (y2 = y1 +1, x2 = x1) }
 P←{ a path from (B.x,B.y) to an entrance vertex in G(V,E)}
 π←arg min_{p∈P}|O(p)| // O(p): a sequence of obstructing blocks on path
                          p in the stock
 map[B.x : B.x+B.w, B.y : B.y+B.h] = 0
 for o in O(π):
     map[o.x : o.x+o.w, o.y : o.y+o.h] = 0
     i←i+1
     schedule[i+1 : ]← schedule[i : ]
     schedule[i].type ← IN
     schedule[i].id ← o.id
     schedule[i].size ← o.size
     schedule[i].time ← B.time
     schedule[i].location ← NA
     insert(schedule, i, map, stock)
 return |O(π)|

}
```

그림 12. 반출 알고리즘 의사코드

4. 실험 및 성능평가

본 논문에서 제안하는 방법의 성능을 검증하기 위해 “random”, “distance”, “route”, “both” 등 4가지 알고리즘을 비교하였다. 그중 random 알고리즘은 이동 가능 그래프에서 나오는 좌표 중 무작위로 반입 위치를 선택하며, distance 알고리즘은 초기 출입구까지 거리 집합을 이용한 조건을 적용하였다. route 알고리즘은 출입구까지 통로를 확보하는 조건을 적용하였다. both 알고리즘은 초기 출입구까지 거리 집합을 이용한 조건과 출입구까지 통로를 확보하는 조건 둘 다를 적용한 반입 알고리즘을 적용하였다. 그리고 그림 12를 동일하게 이용하였다(반출 알고리즘은 최적화 성능에 영향을 미치지 않기 때문이다).

문제 환경을 동일하게 하고 각 알고리즘을 진행하면 반입 불가 블록이 발생하는 순간 적치장 내부 상황이 동일한 조건으로 진행되지 않았다. 그래서 시뮬레이션을 진행하면서 both 알고리즘을 제외한 알고리즘 별로 반입 불가 블록이 발생했을 때 또는 작업이 끝났을 때 간섭 블록의 수가 0이 아니면 바로 이전 task에서 발생한 C_O와 both 알고리즘의 이전 task에서 발생한 C_O을 뺀 차이($diff(C_o)$)를 확인하였고. 그때 both 알고리즘의 반입 불가 블록이 발생했을 때 task와 both 알고리즘을 제외한 각 알고리즘의 반입 불가 블록 발생했을 때 task를 뺀 차이($diff(task)$)를 통해 적치장 활용도를 확인하였다.

표 2는 비교 결과의 평균값과 표준편차를 보여준다(20번씩 시행). 실험 파라미터 중 적치장의 크기를 고정한 반면 블록의 크기, 반입·반출 블록 개수와 출입구 방향을 바꾸어 실험하였다. $diff(C_o)$가 양수이면 비교 알고리즘이 이전 task 기준으로 both 알고리즘보다 C_O가 많은 걸 보여주며, 음수이면 C_O가 작은 걸 보여주기 때문에 수가 클수록 both 알고리즘이 C_O가 적게 나왔다는 결과이므로 좋은 결과를 뜻한다. 그리고 $diff(task)$가 양수이면 both 알고리즘의 task가 반입 불가 블록이 발생할 때까지 더 진행된 걸 보여주며, 음수이면 덜 진행된 걸 보여주기 때문에 수가 클수록 both 알고리즘이 좋은 결과를 뜻한다. 표 3은 표 2

와 다르게 출입구 두 방향을 주면서 출입구가 인접하게 하여 실험을 진행한 결과이다. 표4는 표 3처럼 두 방향은 동일하지만 출입구가 떨어져 있게 하여 실험을 진행한 결과이다. 모든 표에서 both 알고리즘이 $diff(C_o)$와 $diff(task)$가 나은 결과가 나타났다. 표 4에서 distance 알고리즘과 차이가 얼마 나지 않는 것을 보아 해당 조건에서는 both 알고리즘에서 적용된 route 알고리즘이 효과가 크지 않았다는 것을 보인다.

Algorithm / Parameter	random		distance		route	
	$diff(C_o)$	$diff(task)$	$diff(C_o)$	$diff(task)$	$diff(C_o)$	$diff(task)$
Case 1	3.0(2.2)	65.5(22.4)	2.5(2.5)	26.85(35.3)	2.0(4.06)	26.8(36.4)
Case 2	2.0(1.5)	17.7(24.27)	1.7(1.2)	8.0(13.4)	1.6(1.4)	14.7(27.6)
Case 3	1.6(0.9)	18.6(12.3)	NA	NA	2.3(1.9)	7.7(11.1)
Case 4	1.7(0.7)	6.1(2.7)	NA	NA	1.7(1.1)	1.7(4.0)

* Average number of obstruct blocks over 20 trials
* stockyard size : 20x20, entrance direction : 1 direction

Case 1 : block size=2~4, number of insertions/removals blocks=50/50
Case 2 : block size=2~5, number of insertions/removals blocks=50/50
Case 3 : block size=2~4, number of insertions/removals blocks=100/100
Case 4 : block size=2~5, number of insertions/removals blocks=100/100

표 2. 제안 알고리즘과 알고리즘 별 차이 결과(출입구 1방향)

Algorithm / Parameter	random		distance		route	
	$diff(C_o)$	$diff(task)$	$diff(C_o)$	$diff(task)$	$diff(C_o)$	$diff(task)$
Case 1	1.1(1.2)	40.6(26.7)	1.3(2.2)	6.9(15.5)	0.8(1.3)	11.5(20.4)
Case 2	1.6(1.2)	64.8(20.1)	1.0(3.5)	44.4(24.3)	2.0(3.6)	29.5(27.7)

* Average number of obstruct blocks over 20 trials
* stockyard size : 20x20, entrance direction : 2 directions (adjacent)

Case 1 : block size=2~4, number of insertions/removals blocks=50/50
Case 2 : block size=2~4, number of insertions/removals blocks=100/100

표 3. 제안 알고리즘과 알고리즘 별 차이 결과(출입구 2방향)

Algorithm / Parameter	random		distance		route	
	$diff(C_o)$	$diff(task)$	$diff(C_o)$	$diff(task)$	$diff(C_o)$	$diff(task)$
Case 1	3.1(2.0)	35.65(27.2)	0.45(2.8)	1.05(1.77)	3.25(2.6)	23.3(24.6)
Case 2	1.0(1.1)	13.1(11.7)	NA	NA	1.2(1.4)	12.6(11.0)

* Average number of obstruct blocks over 20 trials
* stockyard size : 20x20, entrance direction : 2 directions (away)

Case 1 : block size=2~4, number of insertions/removals blocks=50/50
Case 2 : block size=2~4, number of insertions/removals blocks=100/100

표 4. 제안 알고리즘과 알고리즘 별 차이 결과(출입구 2방향)

5. 결론

본 연구에서는 적치장 효율성을 높이기 위한 반입반출 시스템을 제안하였다. 이를 위해 적치장을 좌표 기반으로 표현하고 블록 이동 가능 그래프와 출입구로부터의 거리 개념을 고안하였다. 이를 바탕으로 선출 및 후출 블록 집합의 특성을 활용하여 최적의 적치 위치를 결정하고 통로를 확보하는 반입알고리즘과 반출 시 최소 간섭 블록의 개수를 계산하는 반출알고리즘을 제시하였다. 실험을 통해 제안된 반입알고리즘의 성능개선 효과를 설명하였지만, 문제 환경에 따라 개선 효과가 변할 수 있음을 확인하였다.

블록 적치 문제는 여전히 많은 과제가 남아있다. 이번에 좌표 기반에서 일정을 활용한 적치 시스템의 기반을 구축했다고 볼 수 있다. 그러나 향후 다음과 같은 연구가 지속해 나가야 한다. 첫째, 직사각형처럼 단순한 모양이 아닌 다양한 모양 적치장이 고려 되어야 한다. 둘째, 두 개 이상의 적치장 사이에서 유동적인 환경 구축이 되어야 한다. 셋째, 일정에 따른 다양한 알고리즘 적용 해봐야한다. 마지막으로 일정을 먼저 최적화는 방법도 고려 되어야 한다.

참고문헌

[1] S. H. Lee, J. O. Kim and I. K. Moon, "Deployment Planning of Blocks from Storage Yards using a Tabu Search Algorithm", in Journal of the Korean Institute of Industrial Engineers, Vol. 37, No.3, pp.198-208, 2011.

[2] M. I. Roh and B. S. Im, "Minimization of the Rearrangement of a Block Stockyard Based on the Genetic Algorithm", in Korean Journal of Computational Design and Engineering, Vol. 16, No.3, pp.207-215, 2011.

[3] M. S. Kim, J. H. Cha and D. Y. Cho, "Determination of Arrangement and Take-out Path in Ship Block Stockyard Considering Available Space and Obstructive Block", in Society for Computational Design and Engineering, pp.433-438, 2013.

[4] J. R. Son, H. W. Suh and B. H. Ha, "A Heuristic Algorithm for Block Storage Planning in Shipbuilding", in Journal of the Society of Naval Architects of Korea, Vol. 51, No. 3, pp.239-245, 2014.

[5] K.K. Cho, K.H. Chung, C. Park, J.C. Park and H. S. Kim, “A spatial scheduling system for block painting process in shipbuilding.” in The International Academy for Production Engineering, CIRP Annals-Manufacturing Technology, Vol. 50, No.1, pp.339-342. 2001.

[6] Y.S. Jung, 2007. “Block transportation management system.” in Master's Thesis, Ulsan National University. 2007.

[7] C.K. Park and J.Y. Seo, “A Case Study on Assembly Block Operations Management at Shipyard”, in Korean Operations Research And Management Society, Vol. 23, No.3, pp.175-186, 2006.

[8] B.H. Ha, J.R. Son, K.K. Cho and B.C. Choi. “A Mathematical Programming Approach for Block Storage Problem in Shipbuilding

Process", Vol. 30, No.3, pp. 99-111, 2013.

[9] J.R. Son and B.H. Ha, "Design of a Block Logistics Operating System in Shipbuilding Industry Based on Axiomatic Design", in the Journal of Society for e-Business Studies, Vol. 19, No.2, pp. 75-93, 2014.

[10] B.W. Nam, K.H. Lee, J.J. Lee and S.H. Mun, "A Study on Selection of Block Stockyard Applying Decision Tree Learning Algorithm" in Journal of the Society of Naval Architects of Korea, Vol. 54, No.5, pp.421-429. 2017.

[영문 요약]

The storage environment where blocks are stored in shipyards is followed by limited selection in a limited environment. There is always a shortage of storage space, and block movement inevitably occurs. Therefore, reducing the storage of blocks and unnecessary movement in shipyards is an area that shipyards always worry about. To solve this problem, it is very important to store blocks in consideration of the schedule for inserting and removing blocks.

In previous studies related to this, the problem of block accumulation was approached by expressing the storage yard by number-based. A limitation of these related studies is that the use of space compared to the size of the block has been reduced. In addition, related studies conducted without a schedule clearly derive limitations that the interference block that occurs is not optimized. In order to improve this existing number-based problem-solving method, the study presented in this paper proposed a method of optimizing block accumulation according to a given block insert and remove schedule by expressing it as a coordinate-based graph. Therefore, we propose a heuristic algorithm that maximizes spatial efficiency and minimizes the number of obstruct blocks. In this paper, the key idea for the problem of storage yard is to create a new graph represented by one vertex for each block, and then apply the exploration algorithm. All movable vertices are obtained. In addition, the distance to the entrance is defined to derive the initial inserting location of the block through the schedule. In addition, the proposed method through simulation results in better spatial efficiency and the number of interference blocks for other

algorithms.

Key words: Shipyard, Block storage optimization, Obstructing block minimization, Block insertion, Block removal, Simulation.

[감사의 글]

학부생 때부터 지금까지 저를 지도해 주셨던 권영근 교수님께 가장 먼저 감사드립니다. 제가 학부생 시절 교수님 수업을 처음 들었던 순간 교수님 밑에서 더 배우고 싶다는 열정과 확신이 들었습니다. 그 결정이 지금 석사를 졸업하는 순간까지 오게 되었습니다. 늘 교수님께서는 제 연구뿐만 아니라 제 삶의 많은 부분에서 지침이 되어 주셨습니다. 교수님께서 지도해 주신 덕분에 제 삶의 과정에서 해야 할 일과 고민에 대하여 스스로 어떻게 나아가야 하며 무엇을 차근차근 준비해 나아가야 하는지 많은 가르침에 감사드립니다. 또한 언제나 교수님께서 아낌없는 조언과 격려를 해주셨기 때문에, 부족한 제 모습에도 스스로 자신감을 가지고 많은 성장해 낼 수 있었습니다. 제가 졸업 이후에도, 앞으로의 제 삶의 모토는 늘 권영근 교수님이라고 말할 수 있습니다. 그만큼 시간이 흐를수록 교수님 같은 어른이 되고 싶은 마음을 간직하게 될 정도로 정말 좋은 가르침을 주셨던 교수님께 다시 한번 감사드립니다. 그리고 학부생 때부터 지금까지 연구실에서 함께해 주었던 이경태 박사과정 연구원께 감사드립니다. 함께 지내왔던 순간들이 연구실에서 좋은 분위기를 마련해주심과 동시에 그동안 많은 프로젝트를 함께하였습니다. 늘 많은 영감이 되어 주셨고, 문제해결 상황마다 가르쳐주신 많은 요령과 함께 해결해 나가는 과정에서 저는 많은 성장을 이뤄낼 수 있었습니다. 또한 언제나 제가 무엇을 해 나가든 제게 아낌없는 성원과 믿음을 보내주셨던 제 부모님 덕분에 제가 무사히 졸업을 해낼 수 있었습니다. 마지막으로 제가 학부생 때부터 석사 과정을 마치기까지 제게 도움을 주셨던 모든 분께 감사드립니다.