Journal of the Korean Institute of Industrial Engineers
Vol. 38, No. 2, pp. 157-172, June 2012.

컨테이너 터미널에서 베이 내 컨테이너의 최적 재정돈을 위한 A* 알고리즘

하병현† · 김상수

부산대학교 산업공학과

A* Algorithm for Optimal Intra-bay Container Pre-marshalling Plan

Byung-Hyun Ha · Sangsu Kim

Dept. of Industrial Engineering, Pusan National University

In most container terminals, containers are piled up and stored in a yard in order to utilize the space efficiently. Hence, it requires unproductive container-handling operations to retrieve a container that is not placed on the top of a container stack. As a result, to streamline container-loading operations by which containers are transferred from a yard to a vessel, it is necessary to pre-marshal (i.e., shuffle in advance) containers in accordance with container-loading plan. We propose A* algorithm to find the optimal container-relocation sequence for the intra-bay container pre-marshalling problem. To work out the heuristic estimate for the proposed A* algorithm, we introduce the container rearrangement problem and obtain the lower bound of the length of the optimal relocation sequence. The performance of the algorithm is validated extensively by the numerical experiments on the problem instances that are given in the previous studies and generated randomly with various parameters.

Keywords: Container Terminal, Pre-marshalling, Intra-bay Operation, Optimization, A* Algorithm

1. 서 론

컨테이너 터미널은 컨테이너의 육로 수송과 해상 수송을 연결하는 접점이다. 선박 또는 트럭 등을 통해 터미널에 들어온 컨테이너는 장치장(storage yard)에 일시적으로 보관되었다가 다른 운송 수단을 통해 터미널 밖으로 나가게 된다. 전 세계적인 컨테이너 물동량 증가에 따라 많은 수의 터미널이 새로 개발되었으며 터미널 간의 경쟁도 치열해지고 있다. 따라서, 경쟁 우위에 서기 위해서는 터미널은 입항하는 컨테이너 선박을 신속하게 출항시킬 수 있도록 생산성을 높여야 한다. 즉, 선박에 실려 온 컨테이너를 내리는 양하 작업과 싣는 적하 작업을 효율적으로 처리하는 것이 관건이다.

효율적인 공간 활용을 위해 장치장에서 컨테이너는 아래에서 위로 쌓여 보관되며, 이러한 장치 방식으로 인해 컨테이너를 꺼내는 반출 작업의 복잡도는 대상 컨테이너가 놓인 위치에 따라 달라진다. 즉, 제일 상단에 장치된 컨테이너는 바로 반출할 수 있으나, 아래쪽에 위치한 컨테이너를 반출하기 위해 서는 그 위에 쌓여있는 컨테이너들을 치워야 하는 비생산적인 작업이 추가된다. 이러한 작업을 컨테이너 재취급(rehandling)이라고 하며 장치장 운영 능력을 떨어뜨리는 주요 요인이 된다. 일반적으로 적하 컨테이너의 반출 순서는 선박에 컨테이너를 싣는 적하 순서를 따른다. 따라서, 컨테이너 장치 상황과 적하 순서의 불일치로 발생하는 재취급 작업은 선박의 출항을 지연시키는 직접적인 요인이 될 수 있다.

본 연구는 장치장에서 적하 대상 컨테이너들을 재취급 작업 없이 쉽게 반출할 수 있도록 미리 정리해두는 컨테이너 재정돈(pre-marshalling) 업무를 대상으로 한다. 이와 같은 컨테이너 재정돈 외에도 효율적인 적하 작업을 고려한 터미널 운영에 대해 많은 연구들이 수행되어 왔다(Kim, 2007; Kim and Kim, 2011). 먼저 컨테이너가 터미널에 도착하면 적하 작업을 예상

이 논문은 부산대학교 자유과제 학술연구비(2년)에 의하여 연구되었음.
† 연락저자 : 하병현 교수, 609-735 부산광역시 금정구 장전 2동 부산대학교 산업공학과, Tel : 051-510-2303, Fax : 051-512-7603, E-mail : bhha@pusan.ac.kr
2012년 3월 12일 접수; 2012년 5월 9일 수정본 접수; 2012년 5월 19일 게재 확정.

하여 장치장에서 컨테이너가 놓일 영역을 지정하고(Kim and Park, 2003; Zhang *et al*., 2003; Won and Kim, 2009; Ku *et al*., 2010; Jeong *et al*., 2012), 지정된 영역 내에서 장치할 위치를 구체적으로 결정한다(Kim and Park, 1996; Kim *et al*., 2000; Kang *et al*., 2006). 또한 컨테이너의 장치 현황을 반영하여 선박에 싣는 순서를 계획하고(Kim *et al*., 1997; Kim *et al*., 2004; Imai *et al*., 2006; Lee, 2011), 장치장에서 적하 컨테이너를 반출할 때 재취급 컨테이너를 옮겨 놓을 위치를 결정한다(Kim and Hong, 2006; Wan *et al*., 2009; Lee and Lee, 2010; Caserta *et al*., 2011). 하지만 컨테이너가 장치장으로 반입되는 시점과 선박에 싣는 적하 순서가 결정되는 시점에는 보통 며칠간의 시차가 존재하며, 컨테이너 수송계획에 예기치 않은 변동이 생기기도 한다. 따라서, 이와 같은 모든 노력에도 불구하고 과도한 재취급이 예상되는 경우 컨테이너 재정돈을 수행하게 된다.

컨테이너의 재정돈은 개별 컨테이너의 장치 위치를 변경하는 일련의 이적(relocation) 작업들로 구성된다. 터미널에서 세로로 쌓인 컨테이너들의 묶음을 스택(stack)이라고 하며, 인접한 스택들이 모여 하나의 베이(bay)를, 연속적으로 위치한 베이는 다시 블록(block)을 이룬다(제 2장의 <Figure 1> 참고). 장치장에서 컨테이너를 취급하는 크레인은 특정 베이에서 정지한 채로 베이 내에 있는 모든 컨테이너를 취급할 수 있다. 따라서 컨테이너 재정돈은 하나의 베이를 대상으로 수행하는 것이 효율적이다. 이와 같은 베이 내 컨테이너 재정돈 계획은 일반적으로 주어진 반출 순서를 기반으로 수립된다. Lee and Hsu(2007)는 베이 내 재정돈 계획을 다품종흐름문제(multi-commodity flow problem)로 형식화하고 최적해를 도출하기 위한 정수계획(integer programming) 모형을 제안하였으며, 제안된 모형을 기반으로 제한된 시간 내에 근사해를 도출할 수 있는 방안을 제시하였다. Lee and Chao(2009)는 근사해를 도출하기 위해 인접해 탐색 기법(neighborhood search)과 이진 정수계획 모형, 세 가지의 휴리스틱 기법으로 이루어진 방법론을 제시하였으며, Caserta and Voβ(2009)는 메타 휴리스틱 기법인 corridor method을 사용하여 재정돈 계획을 도출하였다. Huang and Lin(2012)은 예상되는 재취급 발생 정도에 따라 각 스택을 구분하고 각각 정해진 휴리스틱 기법을 적용하여 이적 순서를 생성하는 방안을 제안하였다. 다음으로, 한 블록 내에서 이루어지는 베이 간 컨테이너의 이적은 보통 선로 위에서 이동하는 크레인(rail-mounted gantry crane)을 사용하는 터미널에서 많이 요구된다. Choe *et al*.(2011)은 한 블록에서 두 대의 크레인을 활용하여 베이 간 컨테이너를 이적하는 방법론을 제안하였다. 마지막으로, 블록 간 컨테이너 이적은 수송 계획에 변동이 생기거나 장치장의 전반적인 부하를 조정할 필요가 있을 때 이루어진다. 블록 간 재정돈 계획의 수립을 위해 Kim and Bae (1998)는 컨테이너 그룹 단위의 계획과 각 그룹 별 상세 계획 수립의 두 단계를 가지는 방법론을 제안하였으며, Huang and Lin(2012)은 베이 내 이적과 유사한 방법론을 확장하여 블록 간 컨테이너 이적 계획 수립 방법론을 제안하였다.

본 연구는 한 베이를 대상으로 하는 재정돈 문제를 다룬다. 한 베이 내에서 컨테이너를 재정돈하는 경우 크레인이 각 컨테이너를 정확하게 집고 내려놓는 것에 가장 많은 노력이 소요된다. 따라서 일반적으로 베이 내 재정돈 계획의 복잡도는 수행되는 이적 작업의 회수로 산정되며, 본 연구에서도 최소 크기의 재정돈 계획, 즉, 최단 길이를 가지는 이적 순서를 도출하는 것을 목적으로 한다. 현재까지 관련 연구에서는 대부분 근사해를 탐색하는 방법론을 제시하였으며, 최적 계획에 대한 연구(Lee and Hsu, 2007)의 경우 과도한 계산 시간으로 현장에 적용하기는 어려웠다. 반면 본 연구에서는 A* 알고리즘(Russell and Norvig, 2003)을 사용하여 상대적으로 짧은 시간에 최적해를 도출하는 방법을 제시한다. 또한, A* 알고리즘을 사용하여 재정돈 문제를 해결하기 위해 추가적으로 컨테이너 재배치 문제(rearrangement problem)를 제시하여 최소 이적 회수의 하한(lower bound)을 구하는데 사용한다. 그리고 기존 연구에서 사용된 문제와 임의로 생성된 문제를 대상으로 수치 실험을 수행하여 본 연구에서 제안하는 방법이 실용적인 크기의 문제에 대하여 최적해를 도출할 수 있음을 보인다.

본 논문의 구성은 다음과 같다. 먼저 제 2장에서는 컨테이너 재정돈 문제와 그 문제의 해결에 사용되는 컨테이너 재배치 문제를 정의한다. 제3장에서는 재정돈 문제를 위한A* 알고리즘을

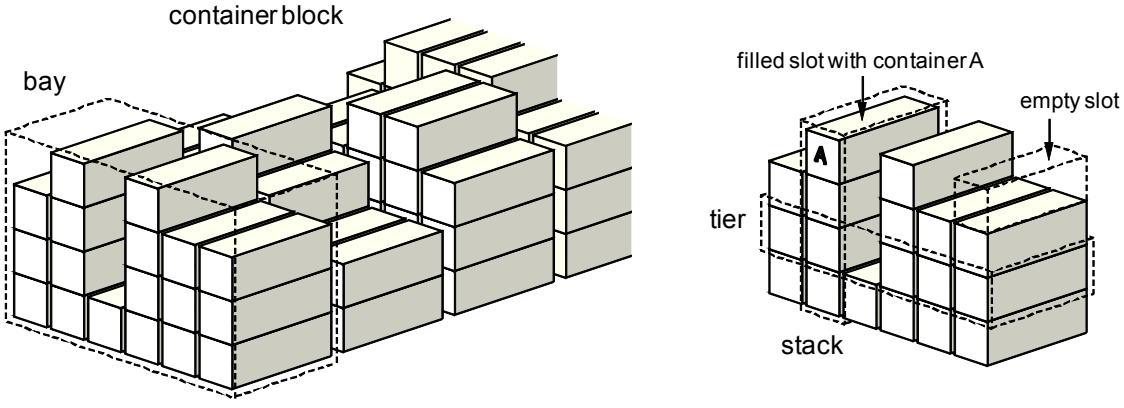

Figure 1. A container block and a bay in a storage yard

제시하고, 제 4장에서는 재배치 문제에 대한 수리 계획 모형과 A* 알고리즘을 사용한 해결 방안을 서술한다. 제 5장에서 수치 실험을 통해 제시한 알고리즘의 성능을 검증하고, 제 6장에서 결론과 추후 연구 과제를 논의한다.

2. 문제 정의

장치장에서 컨테이너는 <Figure 1>과 같이 블록 단위로 보관된다. 각 블록은 여러 개의 베이들로 이루어져있으며, 하나의 베이는 다시 스택들로 구성된다. 각 스택은 컨테이너가 위치한 높이에 따라 단(tier)으로 구분된다. 베이 내에 컨테이너를 장치할 수 있는 공간을 슬롯(slot)이라고 하며, 각 슬롯의 위치는 스택과 단으로 지정된다. 예를 들어 <Figure 1>의 오른쪽에서 주어진 베이를 고려하면, 왼쪽에서 두 번째 스택의 아래에서 네 번째 단에 위치한 슬롯에는 컨테이너 A가 장치되어있다.

2.1 컨테이너 재정돈 문제

본 연구의 대상인 베이 내 컨테이너 재정돈 문제는 기존 연구(Lee and Hsu, 2007; Caserta and Voβ, 2009; Lee and Chao, 2009; Huang and Lin, 2011)에서 다루는 문제와 동일하다. 재정돈 대상 베이는 W개의 스택으로 구성되어있고 H단까지 컨테이너를 쌓을 수 있다. 한 베이에는 동일한 길이의 컨테이너만 장치된다. 이는 컨테이너 터미널에서 베이를 구성하는 일반적인 방식으로, 베이 내에서 어떤 스택으로든 컨테이너를 자유롭게 이적할 수 있다. 장치된 모든 컨테이너는 반출 예정이며, 컨테이너의 반출 순서는 우선순위(priority)를 사용하여 지정한다. 즉, 개별 컨테이너마다 우선순위가 주어져있으며, 높은 우선순위의 컨테이너가 낮은 우선순위보다 먼저 반출된다. 우선순위는 1에서 P까지의 값을 가지는 인덱스(index)를 사용하여 지정하고, 높은 우선순위를 가지는 컨테이너에 작은 값의 인덱스를 부여한다. 둘 이상의 컨테이너가 같은 우선순위를 가질 수 있으며, 우선순위가 동일한 경우 어떤 컨테이너든 먼저 반출할 수 있다. 컨테이너 재정돈 도중 컨테이너 반출 작업은 발생하지 않는다.

<Figure 2>의 (a)는 W = 3이고 H = 3인 베이에 컨테이너가 장치된 예를 보여준다. 장치된 컨테이너는 사각형으로 표시되어있으며 빈 슬롯은 파선으로 구분되어있다. 각 컨테이너에는 컨테이너를 지정하는 알파벳 문자와 함께 우선순위 인덱스가 같이 표시되어있다. 예를 들어 스택 1을 보면, 단 2에는 컨테이너 B가 장치되어 있으며 인덱스는 1로 제일 먼저 반출될 수 있음을 알 수 있다. 컨테이너 D와 F가 가장 큰 인덱스를 가지며 제일 나중에 반출될 컨테이너들이다(P = 3). 스택에 쌓여있는 컨테이너들의 인덱스를 확인하면 재취급 작업의 발생 여부를 알 수 있다. 예를 들어, 컨테이너 D와 F는 아래에 장치된 컨테이너 C와 E보다 큰 인덱스를 가지므로, 재정돈 없이 반출 작업을 시작하는 경우 다른 곳으로 옮겨 두어야 한다. 이와 같은 재취급을 대상이 되는 컨테이너들은 회색으로 표시하였다.

재정돈 문제의 목표는 재취급 작업이 발생하지 않도록 하는 최소 길이의 컨테이너 이적 순서를 찾는 것이다. <Figure 2>의 (b)는 재정돈을 위한 이적 순서의 예를 보여준다. 즉, (a)에서 주어진 베이에서 컨테이너 D와 C를 차례로 스택 3과 스택 1로 이적하고, 다시 D를 스택 2로, 마지막으로 F와 C를 각각 스택 2와 3으로 옮겨 모든 컨테이너를 재취급 없이 반출할 수 있도록 재정돈할 수 있다. 제시된 재정돈 계획의 길이는 5이며, 최단 길이를 가지는 최적 계획이다. 한 번의 이적을 통해 한 스택의 제일 상단에 있는 컨테이너는 다른 스택의 제일 상단으로 옮겨진다. 본 연구에서는 이적 작업을 스택의 쌍으로 표현하고, 재정돈 계획을 나타내는 이적 순서는 일련의 이적들로 표현한다. 예를 들어, <Figure 2>의 (b)에서 제시되어 있는 이적 계획은 (2, 3), (2, 1), (3, 2), (3, 2), (1, 3)이다.

재취급 작업은 베이에 장치된 컨테이너의 인덱스에 따라 결정되며, 인덱스가 같은 컨테이너들의 경우 서로 위치가 바뀌어도 재취급 여부는 달라지지 않는다. 따라서 재정돈 문제에서는 각 슬롯에 위치한 컨테이너의 인덱스만을 사용하여 베이를 표현해도 무방하다. 본 연구에서는 주어진 베이에서 각 슬롯에 위치한 모든 컨테이너를 인덱스로 나타낸 것을 베이의 장치 형태(layout)라 한다. 그리고 최초에 재정돈을 위해 주어진

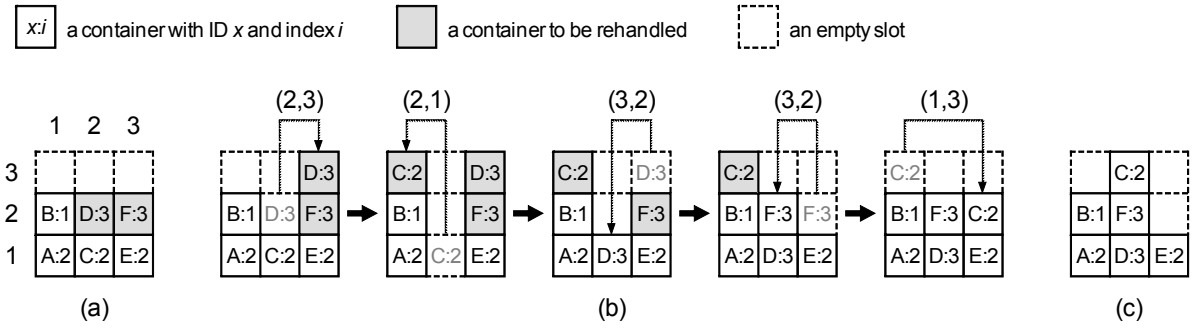

Figure 2. Pre-marshalling of containers in a bay with three stacks and three tiers, (a) an initial layout, (b) a sequence of relocations for pre-marshalling and a final layout, (c) an alternative final layout

베이의 장치형태를 초기형태(initial layout)라고 하고, 재정돈 후 재취급을 발생시키지 않는 장치형태를 최종형태(final layout)라고 한다. 예를 들어 <Figure 2>에서 (a)는 재정돈 문제의 초기형태이며 (b)의 마지막 장치형태는 최종형태를 나타낸다. 따라서 재정돈 문제는 초기형태가 주어졌을 때 최종형태로 바꾸기 위한 최단의 이적 순서를 계획하는 것으로 다시 쓸 수 있다.

초기형태가 주어졌을 때 하나의 이적 순서는 하나의 최종형태를 도출한다. 하지만 그 역이 항상 성립하는 것은 아니다. 즉, 특정 최종형태를 만드는 다른 이적 순서가 존재할 수 있다. 또한 최적 재정돈 계획을 고려할 때, 다른 최종형태를 만드는 같은 크기의 최단 이적 순서가 존재할 수 있다. 예를 들어 <Figure 2>의 (b)에서 주어진 이적 순서를 고려하면, 마지막에 컨테이너 C를 스택 3이 아니라 스택 2로 이적하는 것도, 즉, 마지막의 (1, 3) 대신 (1, 2)의 이적을 수행하는 경우도 최적 재정돈 계획 이며, 그때의 최종형태는 <Figure 2>의 (c)와 같다.

본 연구에서는 A* 알고리즘을 사용하여 재정돈 문제를 해결하며, 이를 위해 최적 이적 순서의 크기에 대한 하한을 도출이 필요하다. 다음 소절에서는 하한을 구하기 위한 컨테이너 재배치 문제를 제시한다.

2.2 컨테이너 재배치 문제

컨테이너 재배치 문제의 목표는 주어진 장치형태에서 일부 컨테이너의 배치를 바꾸어 최종형태를 만드는 것이다. 이때 위치가 바뀌는 컨테이너를 재배치(rearranged) 컨테이너라고 하며, 최종형태로 만들기 위한 재배치 컨테이너의 최소 개수를 찾는 것을 목표로 한다. 재정돈 문제에서는 스택의 최상단에 위치한 컨테이너부터 하나씩 순서대로 이적해서 장치형태를 바꾼다. 반면, 재배치 문제에서는 재배치되는 컨테이너들을 한 번에 모두 집어 올려 목표 위치에 놓을 수 있다고 가정한다. 예를 들어, <Figure 2>에서 주어진 초기형태에 대한 재배치 문제의 최적해는 <Figure 3>과 같이 컨테이너 세 개를 집어 올려 최종형태가 되도록 적절한 슬롯에 다시 장치시키는 것이다. 이 때, 비록 위치가 바뀌지 않더라도 재배치되는 컨테이너 위에 위치한 컨테이너도 (아래에 위치한 컨테이너를 들어올리기 위해서는 위에 있는 컨테이너도 들어야 하므로) 재배치 컨테이너로 본다. 즉, <Figure 3>의 예에서 초기형태에서 스택 2의 단 2에 위치한 인덱스 3을 가지는 컨테이너는 최종형태에서도 동일한 슬롯에 위치가 가능하지만, 아래에 있는 인덱스 2의 컨테이너가 재배치되었으므로 그 컨테이너 역시 재배치된다고 본다.

동일한 장치형태를 입력으로 하는 재정돈 문제와 재배치 문제를 고려하자. 재정돈 문제를 해결하는 이적 순서가 주어진다면, 이적되는 컨테이너들을 재배치 컨테이너로 하여 재배치 문제도 해결할 수 있다. 그리고, 한 번의 이적 작업은 한 컨테이너의 위치만 변경하고, 하나의 컨테이너는 한 번 이상 이적될 수 있다. 따라서, 위와 같이 얻은 재배치 컨테이너의 개수는 재정돈을 위한 이적 순서의 길이보다 작거나 같게 된다. 그러므로, 최종형태를 위한 최소 재배치 컨테이너 개수는 재정돈을 위한 최단 이적 순서의 길이에 대한 하한이 되는 것은 자명하다.

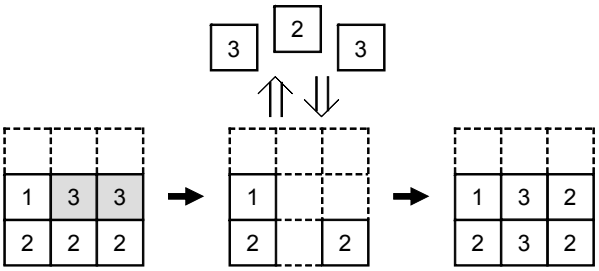

Figure 3. Rearrangement of three containers for a final layout

3. 최적 재정돈 계획

본 절에서는 재정돈 문제를 그래프(graph)에서 최단 경로를 구하는 문제로 형식화하고 A* 알고리즘을 사용하여 최적 이적 순서를 구하는 방법을 제시한다.

장치형태를 나타내는 노드(node)와 이적 작업을 의미하는 아크(arc)로 구성되는 그래프를 고려하자. 이적 작업은 한 장치형태를 다른 장치형태로 변환시킨다. 이에 맞추어 아크는 변환 전후의 장치형태에 대응하는 두 노드를 연결한다. 이와 같은 그래프에서는 두 노드를 잇는 경로(path)가 존재한다면 그 경로 상의 아크들은 시작 노드의 장치형태를 종료 노드의 장치형태로 만드는 이적 순서를 나타내게 된다. 경로 상의 모든 아크 비용의 합을 경로의 길이로 정의하자. 각 아크의 비용을 1이라고 하면 경로의 길이는 해당 이적 순서의 길이가 된다. 이와 같이 그래프를 구성하면, 재정돈 문제는 그래프 상에서 초기형태를 나타내는 시작(start) 노드에서 최종 형태를 나타내는 목표(goal) 노드까지의 최단 경로를 찾는 문제가 된다. 참고로, 시작 노드는 유일하지만 목표 노드는 여러 개가 있을 수 있으며, 시작 노드에서 목표 노드까지 경로가 존재하지 않을 수도 있다.

A* 알고리즘은 방문(visit)하지 않은 노드들 중에서 가장 유망하다고 판단되는 노드를 우선 방문하는 방식으로 최단 경로를 탐색한다. 여기서 유망 노드는 주어진 시작 노드에서부터 목표 노드까지의 최단 경로 상에 있다고 판단되는 노드를 의미하며, 평가 함수(evaluation function) f를 사용하여 결정한다. 즉, 가장 작은 평가 값 $f(n)$을 가지는 노드 n을 우선적으로 방문한다. 평가 함수는 다시 $f(n) = g(n) + h(n)$으로 정의된다. 여기서, g는 깊이 추정 함수로 $g(n)$은 시작 노드에서 n까지의 최단 거리에 대한 추정 값이며, h는 휴리스틱 함수(heuristic function)로 $h(n)$은 n에서부터 목표 노드까지의 최단 거리에 대한 추정 값이다. 모든 노드 n에 대하여 휴리스틱 추정 값이

n에서부터 목표 노드까지의 실제 최단 거리보다 크지 않다면 휴리스틱 함수는 허용가능(admissible)이라고 한다. 목표 노드까지 도달 가능할 때, 휴리스틱 함수가 허용가능이고, 동시에 모든 노드가 한정된 수의 자식 노드를 가지며, 모든 아크의 비용이 0보다 의미 있을 정도로 크다면, A* 알고리즘은 항상 최적해, 즉, 최단 경로를 도출한다(Russell and Norvig, 2003).

본 연구에서는 재정돈 문제를 위한 휴리스틱 추정 값으로 제 2.2절에서 제시한 최소 컨테이너 재배치 개수를 사용한다. 즉, 노드 n의 휴리스틱 추정 값은 n이 나타내는 장치형태를 최종형태로 바꾸기 위해 필요한 최소 재배치 컨테이너 개수이다. 노드 n에서 목표 노드까지의 최단 거리는 n에서의 장치형태를 최종형태로 바꾸기 위한 최소 이적 회수를 의미하므로, 2.2절에서 논의하였듯이 최소 재배치 개수를 휴리스틱 추정 값으로 사용하는 경우 최단 거리의 하한이 된다. 따라서, 이 휴리스틱 함수는 허용가능이다. 또한 한 장치형태에 이적 작업을 적용하여 만들 수 있는 장치형태(자식 노드)의 수는 유한하며 모든 아크의 비용이 1이다. 그러므로, 본 연구에서 사용하는 A* 알고리즘은 최적 재정돈 계획을 도출한다. 최소 재배치 컨테이너 개수를 구하는 방법에 대해서는 제 4장에서 논의한다.

A* 알고리즘에서 휴리스틱 함수가 모든 노드 n과 그 자식 노드 n'에 대하여 $h(n) \leq h(n') + c(n, n')$의 조건을 만족하면 일관적(consistent) 또는 단조적(monotone)이라고 한다. 여기서 $c(n, n')$는 n에서 n'으로 향하는 아크의 비용이다. 재정돈 문제를 고려하면, 노드 n'에서의 휴리스틱 추정 값인 최소 재배치 개수는 n에서의 최소 재배치 개수보다 2이상 줄어들 수 없다. 만일 그렇다고 한다면, n'에서의 재배치 컨테이너들과 n에서 n'으로 이적에 사용된 컨테이너를 합하여 n에서의 재배치 컨테이너로 고려하는 경우, 최종형태를 만들 수 있게 된다. 이는 주어진 n에서의 최소 재배치 컨테이너 개수가 최적이 아니라는 모순을 이끌어낸다. 따라서, 재정돈 문제에서 휴리스틱 함수는 $h(n)-1 \leq h(n')$을 만족하며, 모든 $c(n, n')$이 1이므로 위의 일관성 조건이 성립한다.

A* 알고리즘에서 휴리스틱 함수가 일관적인 경우, 시작 노드에서 우선 방문한 노드까지의 최단 경로가 도출되었음이 보장된다. 즉, 한 번 방문한 노드를 다른 경로를 통해 다시 방문하게 되는 경우 새로운 경로의 길이는 항상 이미 도출된 경로의 길이 이상이 된다. 따라서, 우선 방문한 노드 n까지의 최단 경로의 길이로 계산된 $g(n)$은 시작 노드에서 n까지의 실제 최단 거리가 되며, n을 다시 방문했을 때 n의 자손 노드들의 평가 함수가 다시 계산될 필요가 없다. 이러한 휴리스틱 함수의 일관성은 아래 절차에서 논의된 바와 같이 A* 알고리즘의 수행을 단순하고 효율적으로 만든다.

구체적으로 재정돈 문제를 위한 A* 알고리즘의 절차는 다음과 같다(이 절차는 Nilsson(1998)에서 제시된 알고리즘을 기본으로 서술되었음) :

1. 시작 노드(초기형태)만 들어있는 *OPEN* 리스트와 비어있는 *CLOSED* 리스트를 준비한다.
2. *OPEN*에서 제일 앞에 있는 노드 n을 빼내서(노드 n을 우선 방문) *CLOSED*에 넣는다.
3. 만일 n이 목표 노드(최종형태)면 시작 노드에서 n까지의 최단 경로(최적 이적 순서)를 보고하고 알고리즘을 종료한다.
4. 노드 n이 나타내는 장치형태에서 가능한 이적 작업을 모두 고려하여 자식 노드의 집합 M을 만든다.
5. M에 포함된 각각의 노드 m에 대하여 다음을 수행한다 : m과 동일한 노드가 *OPEN*과 *CLOSED*에 없으면, m을 *OPEN*에 넣고 n에서 m으로 아크를 만든다; m과 동일한 노드가 *OPEN*에 있고 m에서의 평가 값이 기존보다 작으면, 작은 값을 평가 값으로 하고 n에서 m으로 아크를 재설정(reset)한다.
6. *OPEN*이 비어있으면 실패를 보고하고 알고리즘을 종료한다.
7. *OPEN*에 있는 노드들을 평가 값에 따라 오름차순으로 정렬한다. 이때 동일한 최소의 평가 값을 가지는 노드들은 깊이 추정 값에 따라 내림차순으로 정렬된다.
8. 단계 2로 간다.

재정돈 계획을 위해 본 연구에서 사용하는 휴리스틱 함수는 일관적이다. 따라서, 단계 3에서 목표 노드를 방문하는 경우 그때 도출된 경로가 최적이므로 알고리즘을 바로 종료할 수 있다. 그리고, 어떤 노드를 방문했을 때 이미 동일한 장치형태를 가지는 노드가 *CLOSED*에 있는 경우 경로를 개선할 여지가 없으므로 단계 5에서 고려하지 않는다. 반면, *OPEN*에 있는 노드의 경우 더 짧은 경로를 통해 도달할 수 있다. 이러한 경우 단계 5에 기술된 것처럼 최단 경로를 나타내는 아크를 재설정한다. 동일한 최소 평가 값을 가지는 노드들 중 깊이 추정 값이 가장 큰 것을 선택하면 조금 더 일찍 최단 경로를 도출하는 것이 일반적이다. 따라서, 단계 7에서 그에 맞추어 *OPEN*을 정렬한다.

A* 알고리즘을 실행하는 동안 *OPEN*과 *CLOSED*를 사용하여 방문한 노드를 기억해야 하며, 필요한 기억 장소의 크기는 탐색 공간이 커질수록 증가한다. 따라서 저장 공간을 모두 소진하기 전에 최적해를 도출하기 위해서는 탐색 공간의 크기를 줄이는 것이 중요하다. 이를 위해 본 연구에서는 주어진 장치형태를 그대로 사용하는 대신, 장치된 컨테이너의 인덱스에 따라 스택들을 사전순(lexicographical order)으로 정렬한 정규형태(normalized layout)를 사용한다. 즉, 각각의 스택을 아래에서부터 장치된 컨테이너의 인덱스를 사용하여 수열로 표현하고 사전순으로 정렬한다. 이때 빈 슬롯에는 인덱스 0을 가지는 가상의 컨테이너가 장치되어있다고 가정한다. 예를 들어 <Figure 4>의 (a)의 장치형태를 고려하는 경우, 각각의 스택은 0-0-0, 1-4-0, 2-3-1로 표현된다. 이 장치형태는 스택들이 왼쪽에서부터 사전순으로 정렬되어 있으므로 정규형태이다.

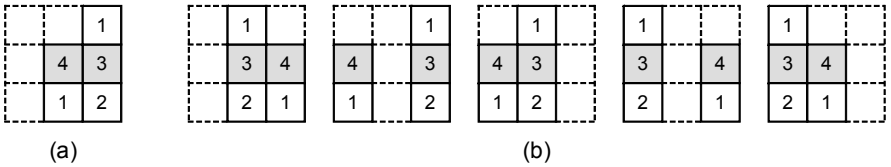

Figure 4. Normalization of layouts, (a) a normalized layout, (b) layouts whose normalized form is the layout in (a)

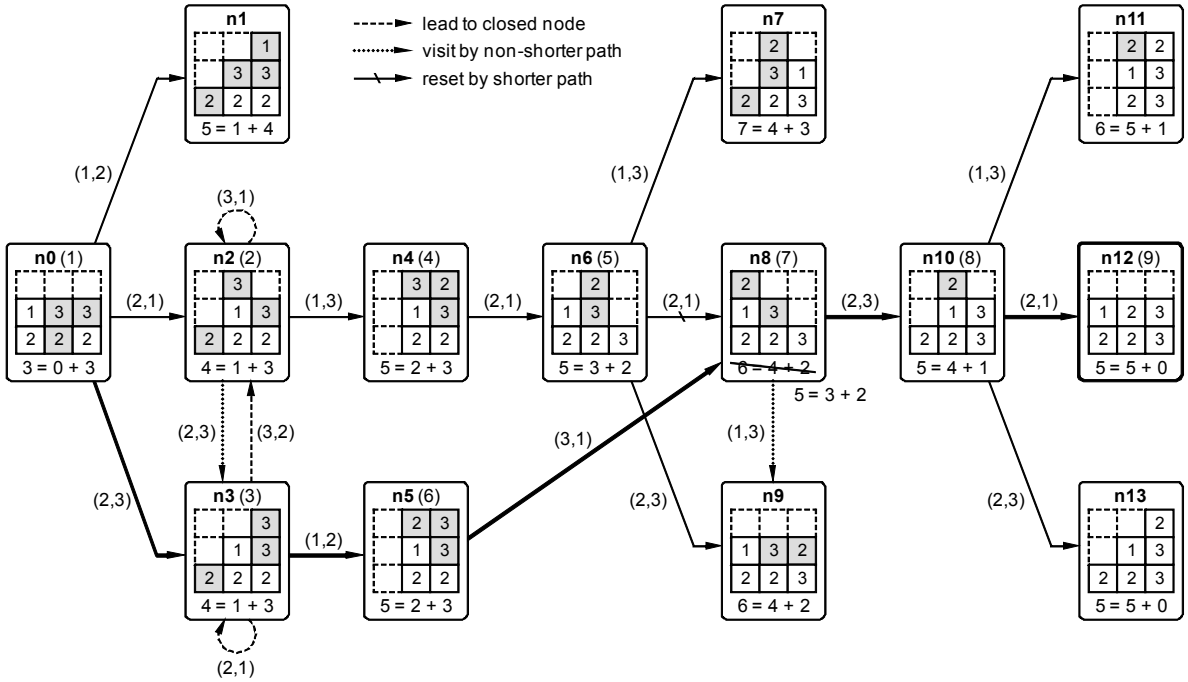

Figure 5. Search of the shortest relocation sequence using A* algorithm for a pre-marshalling problem

이와 같은 방식을 사용하는 경우, (b)의 장치형태들은 모두 (a)와 동일한 정규형태를 가짐을 알 수 있다. 예를 들어 (b)의 두 번째 장치형태의 경우 각각의 스택은 1-4-0, 0-0-0, 2-3-1로 첫 번째와 두 번째 스택이 사전순으로 정렬되어 있지 않다. 이들을 사전순으로 정렬하면, 즉, 첫 번째와 두 번째 스택을 위치를 바꾸면, (a)에서 주어진 정규형태와 같아지게 된다. 따라서 스택의 순서만 다른 장치형태들을 모두 하나로 취급하여 탐색 공간을 줄일 수 있다. 재정돈 문제에서는 목적 값이 이적 순서의 길이로 정의되므로, 서로 다른 장치형태라도 정규형태가 같으면 동일한 최적 값을 가진다. 따라서, 정규형태를 사용하는 경우도 동일한 최적해를 도출할 수 있다. 이와 같은 탐색 공간의 축소 전략은 Kim *et al.*(2000)에서 사용되었다.

<Figure 2>의 (a)에서 주어진 초기형태를 입력 받아 A* 알고리즘을 수행한 결과는 <Figure 5>와 같다. 각 노드는 모서리가 둥근 사각형으로 표시되어 있으며 노드의 이름과 방문 순서가 적혀있다. 예를 들어, 제일 왼쪽에 있는 노드는 초기형태를 나타내며 "n0 (1)"이 적혀있다. 이는 노드의 이름이 n0이며 첫 번째 방문하는 것임을 나타낸다. 각 노드와 자식 노드는 아크를 의미하는 화살표로 연결되어 있으며, 이적 작업이 아크에 표시되어 있다. 예를 들어, n0에서 (1, 2)의 이적을 수행하면, 즉, 스택 1의 최상단 컨테이너를 스택 2로 옮기면, 노드 n1의 장치형태로 변하게 된다. 노드 n1을 보면 알 수 있듯이, 모든 장치형태는 정규형태로 표현되어 있다. 즉, n0의 장치형태에서 (1, 2)의 이적을 수행한 후, 스택 2와 스택 3의 자리를 바꾼 정규형태가 n1에 표시되어 있다. 장치형태 아래에는 평가 값과 깊이 추정 값, 휴리스틱 추정 값이 표시되어 있으며, 휴리스틱 추정 값을 도출하는 최적 재배치 컨테이너 들은 회색으로 표시되어 있다. 예를 들어 n2의 경우, 각 스택의 최상단 컨테이너 세 개를 재배치하면 최종형태를 얻을 수 있음을 알 수 있다. 즉, 휴리스틱 추정 값 h(n2) = 3이며, 평가 값 f(n2) = 4 = 1 + 3 = g(n2) + h(n2)이다.

A* 알고리즘의 탐색은 초기형태를 나타내는 시작 노드 n0을 방문하면서 시작된다. 해당 장치형태에 모든 가능한 이적을 고려하면, 자식 노드인 n1과 n2, n3이 도출된다. 이 때 (1, 2)의 이적뿐 아니라 (1, 3)도 n1과 동일한 노드를 만들게 되지만, 그림에서 따로 표시하지는 않았다. 이와 같은 과정을 마치면 *CLOSED*는 n0을, *OPEN*은 n1, n2, n3을 각각 가지게 된다. 다음 우선 방문 노드는 평가 값이 가장 작으면서 동시에 깊이 추정 값이 최대인 n2와 n3 중 하나가 임의로 선택되게 된다. 이 예에서는 n2가 선택되었다. n2는 목표 노드가 아니므로 *CLOSED*에

추가되며, n2로부터의 가능한 모든 이적을 고려하면 n2, n3, n4의 정규형태를 도출하게 된다. 이때 n2는 이미 *CLOSED*에 있으므로 *OPEN*에 추가하지 않으며, 새로 도출된 n4는 평가 값이 5 = 2 + 3이 되어, 이전의 평가 값보다 커, 역시 *OPEN*에 추가되지 않는다. 이와 같은 사항은 파선과 점선의 아크를 사용하여 나타내었다.

노드 n8은 더 짧은 경로로 *OPEN*에 있는 노드를 다시 방문하는 예를 보여준다. 노드 n8은 n6의 자식으로 처음 생성된다(이때의 깊이 추정 값은 4). 하지만 알고리즘이 진행되면서 n5에서 다시 도달할 수 있게 된다. 결과적으로, 깊이 추정 값이 3으로 줄어들어 평가 값은 5가 되고, 알고리즘의 단계 5에서 기술된 바와 같이 n6에서부터의 아크는 n5로부터의 아크로 재설정된다. 그림에서 n6에서 n8로 향하는 아크가 삭제된 것이 표시되어 있다. 바로 다음, 일곱 번째로 n8을 방문하게 되며, 이때의 깊이 추정 값 3은 시작 노드 n0에서 n8까지의 최단 거리로 확정된다.

알고리즘은 아홉 번째로 n12를 방문하면서 종료하게 된다. 이때의 최적 경로는 굵은 실선으로 표시되어 있으며, 시작 노드에서부터의 최단 거리(최적 이적 순서 길이)는 5임을 알 수 있다. 그림에서 표시된 장치형태가 정규형태임을 고려하여 최적 이적 작업을 기술하면 (2, 3), (2, 1), (3, 2), (3, 2), (1, 3)이 된다. 이는 <Figure 2>의 예에서 주어진 최적 재정돈 계획과 동일하다.

4. 최적 재배치 컨테이너 개수

최적 재정돈 계획을 A* 알고리즘을 사용하여 수립하기 위해서 각 노드의 장치형태에 대해 휴리스틱 추정 값으로 최적 재배치 컨테이너 개수를 도출해야 한다. 본 절에서는 먼저 재배치 컨테이너가 주어졌을 때 최종형태로의 재배치 가능 여부를 판단하는 조건을 제시하고, 이를 사용하여 최적 재배치 컨테이너를 구하기 위한 정수계획 모형을 도출한다. 그리고 역시 A* 알고리즘을 사용하여 최적해를 구하는 방법을 제시한다.

4.1 재배치 가능 조건

한 스택에서 바로 아래 위치한 컨테이너보다 인덱스가 큰 컨테이너와 그 위의 모든 컨테이너를 부적합(misplaced) 컨테이너라 하고, 부적합 컨테이너가 아닌 컨테이너를 적합(well-placed) 컨테이너라고 하자. 예를 들어 <Figure 6>의 (a)에서 제시된 장치형태에서 회색으로 표시된 사각형이 부적합 컨테이너들이다. 장치형태를 최종형태로 바꾸기 위해 부적합 컨테이너는 반드시 재배치되어야 한다. 하지만, 부적합 컨테이너만 위치를 변경하여 최종형태로 만드는 것이 불가능한 경우에는 적합 컨테이너들 중 일부도 추가적으로 재배치해야 한다. 예를 들어, <Figure 6>의 (a)에서는 세 개의 부적합 컨테이너만 재배치하여 최종형태를 만드는 것은 불가능하며, 사선으로 표시된 하나의 적합 컨테이너도 재배치해야 한다. 이와 같이 필요에 따라 재배치되는 적합 컨테이너를 선택 재배치(optionally-rearranged) 컨테이너라고 정의한다. 참고로, 하나의 스택을 고려할 때, 본 연구에서의 부적합 컨테이너 개수는 Lee and Chao(2009)의 mis-overlay index와 동일하다.

주어진 장치형태를 최종형태로 바꾸기 위해서는 재배치 컨테이너들을 모두 들어낸 후 빈 슬롯에 다시 위치시키게 된다. 이와 같이 주어진 재배치 컨테이너들을 위치시킬 수 있는 빈 슬롯을 위치 가능(available) 슬롯이라고 정의한다. 예를 들어 <Figure 6>의 (a)의 예에서와 같이 네 개의 컨테이너를 재배치하는 경우, 가능 슬롯은 (b)에서 파선으로 표시된 것과 같이 모두 일곱 개가 된다.

최종형태에서 가능 슬롯에 장치될 수 있는 재배치 컨테이너는 각 스택에서 재배치되지 않고 남아있는 컨테이너들 중 제일 위에 위치한 컨테이너에 의해 제한을 받는다. 예를 들어 <Figure 6>의 예에서, 스택 1의 경우 인덱스가 2인 컨테이너가 재배치되지 않고 남으므로, 재배치 컨테이너들 중 인덱스가 3인 컨테이너는 최종형태에서 스택 1에 위치할 수 없다. 이와 같은 제한 사항을 나타내기 위해, (b)에서와 같이 가능 슬롯에는 각 스택의 제일 위에 남아있는 컨테이너 인덱스를 부여한다. 스택 내에 재배치되지 않는 컨테이너가 하나도 없는 경우라면 어떤 컨테이너가 장치되어도 재취급을 발생시키지 않으므로 가능 슬롯에 최대 인덱스 P를 부여한다.

특정 장치형태에 대하여 가능 슬롯의 개수와 인덱스는 주어진 재배치 컨테이너들에 의해 결정된다. 장치형태와 재배치 컨테이너가 주어졌을 때, $i = 1, \cdots, P$에 대하여 인덱스 i를 가지는 가능 슬롯의 개수와 재배치 컨테이너의 개수를 각각 α^i와 β^i라 하자.

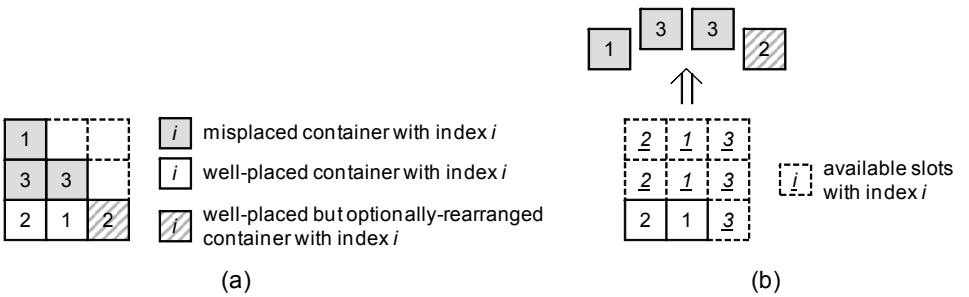

Figure 6. Rearranged containers and available slots, (a) misplaced, well-placed, and optionally-rearranged containers, (b) available slots where four rearranged containers can be located

예를 들어 <Figure 6>의 경우 가능 슬롯의 개수는 $\alpha^1 = 2$, $\alpha^2 = 2$, $\alpha^3 = 3$이며, 재배치 컨테이너 개수는 $\beta^1 = 1$, $\beta^2 = 1$, $\beta^3 = 2$이다. 이와 같은 경우, 재배치 컨테이너의 위치를 변경해 최종형태로 만들 수 있는지 여부는, 즉, 주어진 재배치 컨테이너가 가능해(feasible solution)인지는 다음의 보조정리 1에 따라 판단할 수 있다.

보조정리 1 : 인덱스 i를 가지는 가능 슬롯과 재배치 컨테이너의 개수가 각각 α^i와 β^i로 주어졌을 때, 최종형태로 재배치 가능한 경우의 필요충분조건은 다음과 같다 :

$$\sum_{l=i}^{P} \alpha^l \geq \sum_{l=i}^{P} \beta^l \quad \text{for} \quad i = 1, \cdots, P \tag{1}$$

증명 : 먼저 최종형태가 가능하다고 하자. 즉, 주어진 재배치 컨테이너의 위치를 변경하여 최종형태가 만들어진다. 최종형태에서는 재취급 작업이 발생하지 않으므로, 인덱스가 i 이상인 재배치 컨테이너는 i보다 작은 인덱스를 가진 가능 슬롯에는 장치되지 않는다. 즉, i 이상의 인덱스를 가진 가능 슬롯들은 해당 컨테이너들이 모두 장치될 수 있을 만큼 충분하다. 이는 곧 식 (1)의 조건이며, 따라서 이 조건은 만족한다.

다음으로 식 (1)의 조건이 만족함을 가정하고, 최종형태가 가능함을 보인다. 이를 위해, 재배치 컨테이너를 인덱스가 가장 큰 것부터 하나씩 가능 슬롯에 장치시키는 재배치 방식을 고려하자. 이 때 인덱스가 크고 아래 단에 위치한 가능 슬롯에서부터 컨테이너를 장치한다. 식 (1)의 조건이 성립하므로, $\alpha^P \geq \beta^P$이 만족한다. 즉, 인덱스 P를 가지는 가능 슬롯은 인덱스 P의 컨테이너보다 많다. 따라서, 인덱스가 P인 컨테이너들을 위의 방식으로 재배치하면 재취급 작업을 발생시키지 않는다. 인덱스 i까지의 컨테이너를 재취급 작업을 발생시키지 않도록 위의 방식으로 재배치하였다고 가정하자. 이 경우 인덱스가 i 이상인 가능 슬롯은 $\left(\sum_{l=i}^{P} \alpha^l - \sum_{l=i}^{P} \beta^l\right)$개 남게 되며, 인덱스가 $(i-1)$인 가능 슬롯이 α^{i-1}개 있으므로, 인덱스가 $(i-1)$ 인 컨테이너를 장치시킬 수 있는 가능 슬롯은 모두 $\left(\sum_{l=i-1}^{P} \alpha^l - \sum_{l=i}^{P} \beta^l\right)$개 존재한다. 식 (1)이 성립한다고 가정하였으므로, 해당 가능 슬롯의 개수는 아래 식 (2)와 같이 적어도 β^{i-1} 이상이다.

$$\sum_{l=i-1}^{P} \alpha^l - \sum_{l=i}^{P} \beta^l \geq \sum_{l=i-1}^{P} \beta^l - \sum_{l=i}^{P} \beta^l = \beta^{i-1} \tag{2}$$

즉, 인덱스가 $(i-1)$인 컨테이너 β^{i-1}개를 모두 재취급 작업이 없도록 장치시킬 수 있다. 결과적으로 귀납에 의해 모든 재배치 컨테이너를 재취급 작업이 발생하지 않도록 재배치하여 최종형태를 만들 수 있다. □

4.2 수리계획 모형

스택 j에서의 선택 재배치 컨테이너 개수 x_j를 결정변수로 고려하자. 즉, 최종형태를 만들기 위해 스택에 있는 부적합 컨테이너와 함께 상단에 위치한 적합 컨테이너 x_j개가 재배치된다. 주어진 장치형태에서 스택 j에 위치한 적합 컨테이너 개수를 n_j라고 하면, $0 \leq x_j \leq n_j$의 범위를 가진다. 재배치 문제의 목적은 재배치되는 적합 컨테이너의 총 개수 $\sum_{j=1}^{W} x_j$를 최소화하는 것이다.

최종형태에서 특정 스택에 재배치 될 수 있는 컨테이너는 선택 재배치 컨테이너 개수에 의해 결정된다. 이를 나타내기 위해 본 연구에서는, 인덱스 i의 재배치 컨테이너가 최종형태에서 스택 j에 위치할 수 있기 위해, 스택 j에서 재배치해야 하는 적합 컨테이너의 최소 개수 $r(i, j)$를 도입한다. 예를 들어, <Figure 7>에서 주어진 장치형태를 고려하자. 부적합 컨테이너는 회색으로 표시되어 있으며, 이 때의 $r(i, j)$는 <Table 1>과 같다. 이 표를 통해 만일 인덱스가 3인 재배치 컨테이너가 스택 1에 장치될 수 있기 위해서는 그 스택에서 둘 이상의 적합 컨테이너가 재배치되어야 하는 것을 알 수 있다($x_1 \geq r(3,1) = 2$).

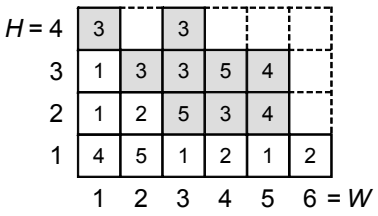

Figure 7. An example of a layout of 18 containers in 6 stacks and 4 tiers ($P = 5$)

Table 1. The minimum number of well-placed containers in stack j, $r(i, j)$, which must be rearranged when a rearranged container with index i can be located in stack j at the final layout, considering the layout in <Figure 7>

	j					
i	1	2	3	4	5	6
1	0	0	0	0	0	0
2	2	0	1	0	1	0
3	2	1	1	1	1	1
4	2	1	1	1	1	1
5	3	1	1	1	1	1

스택 j에서의 선택 재배치 컨테이너 개수 x_j 이하의 k에 대해서는 1의 값을 가지고 그렇지 않은 경우 0의 값을 갖는 보조 변수 y_{jk}를 도입하자. 즉, x_j와 y_{jk}는 다음과 같은 관계를 가진다:

$$y_{jk} = \begin{cases} 1, & \text{if } k \le x_j \\ 0, & \text{otherwise} \end{cases} \quad \text{for } k = 0, \cdots, n_j \tag{3}$$

예를 들어, <Figure 7>에서 주어진 장치형태에서 x_4 = 2인 경우, 즉, 세 개의 적합 컨테이너중 위의 두 개가 재배치되는 경우, y_{40} = 1, y_{41} = 1, y_{42} = 1, y_{43} = 0이 된다. 주어진 x_j에 대하여, y_{j0}는 항상 1의 값을 가지고, $y_{j1} = \cdots = y_{jx_j} = 1$이며 $y_{j(x_j+1)} = \cdots = y_{jn_j} = 0$이 되므로, $x_j = \sum_{k=1}^{n_j} y_{jk}$가 성립한다. 추가적으로, c^i를 인덱스 i를 가지는 부적합 컨테이너의 개수라 정의하면(<Figure 7>의 예에서는 c^1 = 0, c^2 = 0, c^3 = 5, c^4 = 2, c^5 = 2), 보조정리 1의 조건 (1)은 y_{jk}와 $r(i,j)$를 사용하여 다음의 정리 2에 의해 구체화된다(정리 2에 대한 증명은 부록에 실었다).

정리 2 : 각 스택에서의 선택 재배치 컨테이너 개수가 주어졌을 때, 최종형태가 가능한 필요충분조건은 다음과 같다.

$$\sum_{j=1}^{W} (H - n_j + r(i,j)) y_{jr(i,j)} \ge \sum_{l=i}^{P} c^l \quad \text{for } i = 1, \cdots, P$$

선택 재배치 컨테이너 개수 x_j가 주어졌을 때, 인덱스 i에 대하여 식 (4)의 좌변은 인덱스가 i 이하인 재배치 컨테이너를 위치시킬 수 있는 가능 슬롯(즉, 인덱스 i 이상인 가능 슬롯)의 개수에서 가능 슬롯의 위치에 장치되어있던 인덱스 i 이상인 선택 재배치 컨테이너의 개수를 뺀 값을 의미하며, 우변은 인덱스가 i 이상인 부적합 컨테이너의 개수를 의미한다. 따라서, 식 (4)는 식 (1)의 양변에서 각각 인덱스가 i 이상인 선택 재배치 컨테이너의 개수를 뺀 형태임을 알 수 있다.

결과적으로, 스택 j에서의 선택 재배치 컨테이너의 개수 x_j를 결정변수로 하는 재배치 문제를 (P)라고 하면 정수계획 모형은 다음과 같다:

$$\min. \sum_{j=1}^{W} x_j \tag{5}$$

$$\text{s.t.} \quad x_j = \sum_{k=1}^{n_j} y_{jk} \quad \text{for } j = 1, \cdots, W \tag{6}$$

$$y_{j(k-1)} \ge y_{jk} \quad \text{for } k = 1, \cdots, n_j \quad \text{for } j = 1, \cdots, W \tag{7}$$

$$\sum_{j=1}^{W} (H - n_j + r(i,j)) y_{jr(i,j)} \ge \sum_{l=i}^{P} c^l \quad \text{for } j = 1, \cdots, P \tag{8}$$

$$y_{jk} = 0 \text{ or } 1 \quad \text{for } k = 0, \cdots, n_j \quad \text{for } j = 1, \cdots, W \tag{9}$$

식 (5)의 목적함수는 각 스택에서 재배치되는 적합 컨테이너 개수의 합으로, 재배치 컨테이너의 총 개수는 $\left(\sum_{j=1}^{W} x_j + \sum_{i=1}^{P} c^i\right)$가 된다. 식 (6)과 식 (7)은 선택 재배치 컨테이너 개수에 맞추어 y_{jk}의 값을 제한하며, 식 (8)은 정리 2에서 제시된 최종형태가 되기 위한 필요충분조건이다.

4.3 재배치 문제를 위한 A* 알고리즘

최소 재배치 컨테이너 개수를 구하기 위한 A* 알고리즘에서, 노드는 재배치 계획, 즉, 각 스택에서의 선택 재배치 컨테이너 개수를 나타낸다. 시작 노드는 모든 스택에서 선택 재배치 컨테이너가 하나도 없는 경우에 해당한다. 특정 노드가 나타내는 재배치 계획으로 최종형태를 만들 수 있는지 여부는 정리 2의 조건으로 판단할 수 있으며, 그 조건을 만족시키는 목표 노드를 방문하면 그 계획이 최적해, 즉, 최소 개수의 선택 재배치 컨테이너를 가지는 계획이 되고 알고리즘은 종료한다. 특정 노드를 방문하였을 때 해당 재배치 계획으로 최종형태를 만들지 못하는 경우, 각 스택에서의 선택 재배치 컨테이너 개수를 증가시켜 자식 노드들을 생성한다. 부모 노드와 자식 노드를 연결하는 아크는 증가된 재배치 컨테이너의 개수만큼의 비용을 가진다. 따라서 시작 노드에서 목표 노드까지의 최단 거리는 최종형태를 만들기 위한 최소 재배치 컨테이너 개수가 된다.

최종형태의 조건인 식 (8)을 고려하면, $r(i,j)$의 값은 인덱스 i에 대하여 단조증가하므로, 큰 인덱스에 해당하는 조건을 만족시키기 위해서는 그보다 작은 인덱스에 해당하는 조건보다 더 크거나 같은 x_j가 요구된다. 이는 곧, 만족되지 않는 조건들 중 가장 큰 인덱스에 해당하는 조건을 만족하도록 x_j를 증가시켰을 때, 즉, 더 많은 적합 컨테이너를 재배치할 때, 다른 조건들도 따라서 만족될 수 있음을 의미한다. 그러므로, 본 연구에서는 주어진 노드에 대하여 식 (8)이 만족하지 않는 가장 큰 인덱스에 대하여, 그 조건을 만족시킬 수 있도록 x_j를 증가시켜 자식 노드들을 만든다. 구체적으로, 이와 같은 가장 큰 인덱스를 $\hat{i}$이라고 하자. 스택 j에 대해서 만일 $x_j < r(\hat{i}, j)$라면 $y_{jr(\hat{i},j)}$ = 0이므로 $y_{jr(\hat{i},j)}$가 1이 되도록 x_j를 $r(\hat{i}, j)$로 만들면 식 (8)의 좌변을 증가시킬 수 있다. 따라서, $x_{\hat{j}} < r(\hat{i}, \hat{j})$인 각각의 $\hat{j}$에 대하여 다음의 재배치 계획을 가지는 자식 노드를 만든다:

$$x_j' = \begin{cases} r(\hat{i}, \hat{j}), & \text{if } j = \hat{j} \\ x_j, & \text{otherwise} \end{cases} \quad \text{for } j = 1, \cdots, W \tag{10}$$

<Figure 7>에서 주어진 장치형태에 대해 최소 재배치 컨테이너 개수를 구하는 A* 알고리즘의 탐색 절차는 <Figure 8>과 같다. 각 노드의 장치형태에서 선택 재배치 컨테이너는 사선으로 표시되어 있으며, 그에 해당하는 x_j는 장치형태 아래 표시되어 있다. 각 아크에는 비용, 즉, 재배치 컨테이너 개수의 증가분이 표시되어 있다. 시작 노드인 n0를 고려하면 1이 아닌 모든 인덱스는 재배치 가능 조건인 식 (8)을 만족시키지 못하여 최대 인덱스 $\hat{i}$은 5가 된다. 즉, 주어진 재배치 계획에서는 인덱스가 5인 재배치 컨테이너를 놓을 충분한 수의 가능 스택이 존재하지 않는다. 이 경우 각각의 가능한 스택(이 경우는 모든 스택) $\hat{j}$에 대하여 $x_{\hat{j}}$을 충분히 증가시켜, 즉, 인덱스가 5인 재배치 컨테이너를 장치시킬 수 있을 만큼 선택 재배치 컨테이너의 개수를

$r(5, \hat{j})$로 늘려, n1부터 n6까지의 자식 노드를 만든다. 예를 들어 $\hat{j}$ = 1일 때, 즉, 스택 1을 고려하면, $r(5, 1) = 3$이므로 n1의 x_1은 3이 되며, n0에서 n1을 잇는 아크 비용은 3이 된다. 각 노드에는 이름과 방문 순서, 최대 인덱스 $\hat{i}$를 표시하였다.

각 노드의 휴리스틱 추정 값은 재배치 문제 (P)를 각각 하나의 인덱스에 대해서만 재배치 가능성을 고려하는 문제로 분해하여 구한다. 이때, 결정변수는 선형으로 완화한 $\hat{y}_{ik}$를 고려한다. 구체적으로, $j = 1, \cdots, W$에 대하여 선택 재배치 컨테이너 개수가 x_j인 특정 노드를 고려하자. 식 (8)이 만족하지 않는 인덱스의 집합을 I라 하고, 각각의 $i \in I$에 대하여 J_i를 인덱스가 i인 재배치 컨테이너를 장치시킬 수 있는 스택(가능 슬롯의 인덱스가 i 이상인 스택)의 집합으로 정의하자. 즉, $x_j \geq r(i, j)$를 만족시키는 j가 J_i의 원소가 된다. 그리고 $J_i' = \{1, \cdots W\} \backslash J_i$로 정의하자. 그러면, 각각의 $i \in I$에 대하여, 분해된 문제 (P_i)는 다음과 같은 선형계획(linear programming) 모형으로 정의된다:

$$\min. \quad z_i = \sum_{j \in J_i'} (r(i,j) - x_j) \hat{y}_{ir(i,j)} \tag{11}$$

$$\text{s.t.} \quad \sum_{j \in J_i'} (H - n_j + r(i,j)) \hat{y}_{jr(i,j)} \geq \sum_{l=i}^{P} c^l - \sum_{j \in J_i} (H - n_j + r(i,j)) \tag{12}$$

$$0 \leq \hat{y}_{ir(i,j)} \leq 1 \quad \text{for } j \in J_i' \tag{13}$$

각각의 스택 $j \in J_i'$에 대하여 $\hat{y}_{ir(i,j)}$가 이 모형의 결정변수이며, 목적 값 z_i는 주어진 계획 대비, 인덱스가 i인 재배치 컨테이너를 모두 재취급을 발생하지 않도록 장치하기 위해, 추가적으로 재배치해야 하는 적합 컨테이너의 개수의 하한을 의미한다. 참고로 선형계획 문제로 완화하였으므로 최적 z_i는 정수가 아닐 수 있다.

각각의 $i \in I$에 대하여 (P_i)의 최적 목적 값을 z_i^*라 할 때, $\tilde{z} = \max_{i \in I} z_i^*$로 정의하자. 즉, $\tilde{z}$는 각각의 인덱스 하나씩만 고려해서 구한 최종형태를 위해 추가적으로 재배치해야 하는 최소 적합 컨테이너 개수의 하한 중 최댓값을 나타낸다. 이 경우, $\left(\sum_{j=1}^{W} x_j + \tilde{z}\right)$는 원문제 (P)의 최적 값인 최소 선택 재배치 컨테이너 개수의 하한이 되는 것은 자명하다. $\sum_{j=1}^{W} x_j$는 주어진 노드의 깊이 추정 값이므로, 따라서, $\tilde{z}$는 허용가능인 휴리스틱 추정 값이다. 또한, 각각의 $i \in I$에 대하여, 주어진 부모와 자식 노드에서의 이 모형의 최적 목적 값을 고려하면, 두 노드를 잇는 아크의 비용인 추가적으로 재배치되는 컨테이너 개수 보다 최적 목적 값이 더 많이 줄어들 수 없다. 따라서, $\tilde{z}$은 일관적인 추정 값이다.

선형계획 (P_i)를 효과적으로 풀이하기 위해서 선형으로 완화한 0-1 배낭문제(knapsack problem)의 최적 값을 구하는 방법(Winston, 2004)을 적용할 수 있다. 즉, $j \in J_i'$에 대하여, 0인 $\hat{y}_{ir(i,j)}$가 1이 되면, 목적 값은 $(r(i, j) - x_j)$ 만큼 증가하고, 식

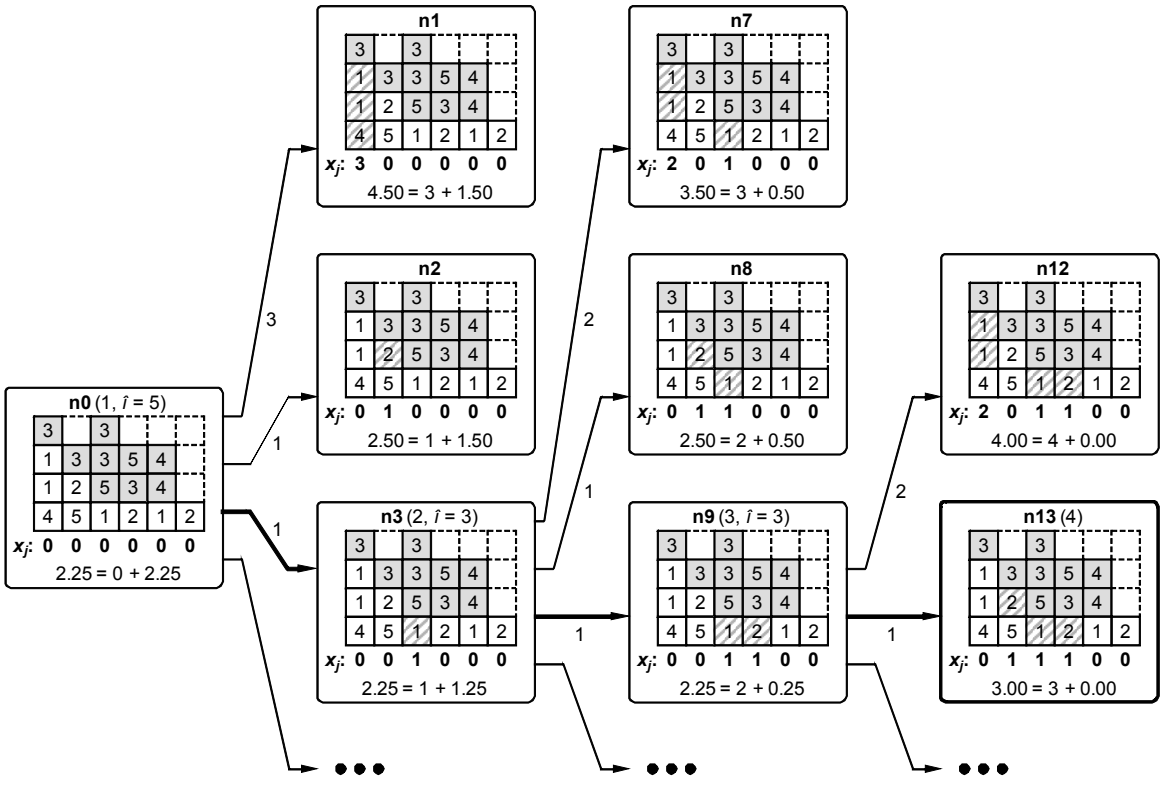

Figure 8. Search of the minimum number of reassigned containers using A* algorithm for a reassignment problem

(12)의 좌변은 $(H-n_j+r(i,j))$ 만큼 증가한다. 따라서 식 (12)를 만족시키기 위해 0인 $\hat{y}_{ir(i,j)}$를 1로 만들어서 얻을 수 있는 효과는 $(H-n_j+r(i,j))/(r(i,j)-x_j)$가 된다. 이러한 효과가 높은 순으로 j를 고려하면서, 식 (12)를 만족시킬 수 있을 때까지 $\hat{y}_{ir(i,j)}$를 증가시키는 경우 (P_l)의 최적해를 찾을 수 있다. <Figure 8>의 각 노드 아래쪽에는 평가 값과 깊이 추정 값, 휴리스틱 추정 값이 표시되어 있다. 예를 들어 n2의 경우, $I=\{2,3,4\}$이고, $z_2=0.25$, $z_3=1.5$, $z_4=0.25$가 되어, 휴리스틱 추정 값 $\tilde{z}$는 1.5가 된다. 그리고, 깊이 추정 값은 1이므로 평가 값은 2.5이다.

재배치 문제를 위한 A* 알고리즘의 절차는 재정돈 문제를 위한 절차와 크게 다르지 않으므로 따로 기술하지 않는다.

5. 수치 실험

본 연구에서 제시한 최적화 알고리즘의 성능을 평가하기 위해, 먼저 기존 연구에서 사용된 문제를 풀고 결과를 비교한다. 다음으로, 다양한 설정의 사용하여 임의로 생성한 문제를 풀어 성능을 분석한다. 알고리즘은 Python을 사용하여 구현하였으며, Intel Core i3-2100과 4GB RAM의 하드웨어와 Windows 7(64bit) 운영체제 상에서 CPython 2.7.2를 사용하여 실험하였다.

5.1 기존 연구 결과와 비교

<Figure 9>는 기존 연구에서 성능 비교의 의미가 있다고 파악된 세 개의 문제, P1, P2, P3를 나타낸다. <Table 2>는 각각에 대하여 문제의 크기와 기존 연구의 결과와 본 연구의 실험 결과를 보여준다. 문제의 크기는 베이를 구성하는 단의 수(H)와 스택의 수(W), 최대 인덱스(P), 장치된 컨테이너의 수로 나타내었다. 그리고 각각의 문제에 대해, 기존 연구의 결과에 대하여 발표된 문헌(Study), 최적화 방법론의 사용 여부 (Method), 최종형태로 만드는 이적 순서의 크기(Size of Plan), 초 단위의 계산 시간(CPU Time)을 기술하였다. 기존 연구의 계산 시간은 문헌에서 보고된 시간을 사용하였으며, Huang and Lin(2011)의 경우는 계산 시간이 보고되지 않아 비워두었다.

본 연구의 결과는 이적 순서의 크기와 계산 시간, 그리고 최적해를 찾기 위해 방문한 노드 수(Visits)를 사용하여 기술하였다. P1의 경우 노드를 10 번 방문하여 크기가 9인 최적 이적 순서를 찾으며, 이는 최소한의 노드 방문을 통해 최적해를 찾았음을 의미한다. 역시 P2의 경우도 최적해를 찾는데 거의 시간이 걸리지 않았으며, 상대적으로 큰 문제인 P3의 경우도 2초가 소요되지 않는 것을 알 수 있다. <Table 3>은 각각의 문제에 대해 실험을 통해 찾은 최적 이적 순서를 보여준다.

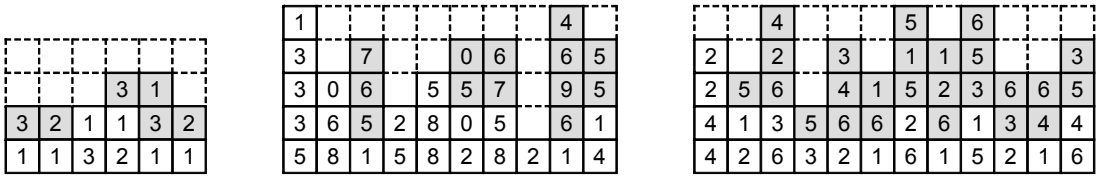

Figure 9. Problem instances in the previous studies, (a) P1, (b) P2, (c) P3

Table 2. Comparison of the experimental results against the previous studies (CPU Time in sec.)

	H	W	P	Number of Containers	Previous Results				Our Results		
					Study	Method	Size of Plan	CPU Time	Size of Plan	CPU Time	Visits
P1	4	06	03	14	Lee and Hsu(2007)	Optimal	9	6,802	9	< 0.1	10
					Huang and Lin(2011)	Heuristic	9	-			
P2	5	10	10	35	Lee and Chao(2009)	Heuristic	31	5	15	< 0.1	24
					Caserta and Voβ(2009)	Heuristic	19	20			
P3	5	12	06	45	Lee and Hsu(2007)	Heuristic	47	318	30	< 1.9	1,369
					Huang and Lin(2011)	Heuristic	35	-			

Table 3. Optimal solutions of the problem instances in <Figure 9>

	Optimal Solution
P1	(5, 3), (1, 6), (1, 3), (4, 1), (6, 1), (2, 1), (6, 1), (2, 4), (5, 2)
P2	(9, 5), (6, 5), (8, 4), (2, 4), (9, 2), (9, 8), (3, 8), (9, 2), (7, 2), (7, 8), (3, 8), (3, 7), (6, 8), (10, 7), (10, 7)
P3	(8, 1), (4, 11), (12, 4), (8, 4), (6, 4), (6, 5), (6, 4), (10, 6), (9, 6), (2, 6), (9, 6), (12, 6), (3, 12), (10, 12), (10, 12), (8, 10), (8, 2), (11, 10), (11, 8), (5, 8), (7, 10), (7, 2), (7, 10), (3, 7), (3, 8), (9, 3), (5, 3), (5, 10), (5, 8), (11, 8)

5.2 임의로 생성된 문제

다양한 문제에 대해 제시된 알고리즘의 성능을 검증하기 위해 설정을 달리하여 임의로 문제를 생성하고 실험을 수행하였다. 문제의 생성을 위해 단수(H)는 4, 6, 8을 사용하였으며, 스택의 개수(W)는 6, 8, 10, 12를 사용하였다. 동일한 인덱스를 가지는 컨테이너의 비율이 계산 시간에 미치는 영향을 파악하기 위해 최대 인덱스(P)의 값으로 5, 10, 15, 20, ∞를 사용하였다. 여기서 P = ∞는 모든 컨테이너가 다른 인덱스를 가지는 것을 의미한다. 그리고 한 베이에 장치할 수 있는 컨테이너의 총 개수 대비 실제 장치된 개수의 비율을 나타내는 장치율 u는 40%, 50%, 60%, 70%, 80%를 사용하였다. 참고로, 80%까지의 장치율을 사용하는 경우 모든 문제에 대하여 최종형태로 만들 수 있는 이적 순서가 존재한다.

문제를 생성하기 위해, 주어진 설정에 대하여 WHu를 계산한 후 반올림하여 장치될 컨테이너 개수를 정했다. P = ∞인 경우를 제외하면, 각각의 컨테이너는 동일한 확률로 1에서 P까지의 인덱스 중 하나를 가지게 하였다. 따라서, 컨테이너가 많지 않은 경우 P의 변화가 문제의 형태에 큰 영향을 미치지 않는다. 베이에 컨테이너가 장치된 형태를 만들기 위해, 모든 슬롯의 집합에서 컨테이너 개수만큼 슬롯을 하나씩 비복원 추출하여 컨테이너를 배정하였으며, 최종적으로 각각의 스택에서 장치된 컨테이너 사이에 위치하는 빈 슬롯을 제거하여 유효한 장치형태를 만들었다.

실험은 각각의 설정의 조합에 대하여 30개의 문제를 임의로 만들어서 수행하였으며, 그 결과는 <Table 4>로 정리하였다. Solvable의 열은 생성된 30개의 문제들 중 최적해를 구할 수 있었던 회수(S)와 그 비율(%)을 나타낸다. 주어진 기억 장소를 다 소모할 때까지 탐색을 완료하지 못하면 최적해를 구하지 못하고 종료하게 된다. 옅은 회색으로 표시된 칸은 일부 문제에 대해서만 해를 도출할 수 있었던 설정을 나타내며, 짙은 회색은 생성된 30개의 문제를 하나도 해결하지 못한 경우를 나타낸다. 최적해를 구하는데 성공한 문제들에 대해서만 계산에 소요된 시간의 최대 값(MAX)과 평균(AVG)을 초 단위로 표시하였다.

먼저, <Figure 9>에서 주어진 문제를 풀이한 결과가 임의의 문제에 대한 실험 결과와 크게 배치되지 않는 것을 알 수 있다. 즉, P1의 경우 장치율이 약 60%이고 H와 W가 각각 4와 6이므로 <Table 4>에서 주어진 결과와 일치하며, P2와 P3의 경우 각각 장치율이 70%와 75%이고 H = 5이므로 <Table 4>에서 유사한 설정의 결과와 비교하면 역시 결과가 크게 다르지 않음을 확인할 수 있다. 따라서, 본 연구에서 제시한 방법론이 기존 연구의 문제들에 대해서만 특별히 좋은 결과를 도출하는 것이 아님을 알 수 있다.

실험 결과를 전체적으로 살펴보면, 네 단의 장치형태에 대해서는 모든 경우 해를 구할 수 있었던 것을 알 수 있으나, 단수와 스택의 수, 그리고 장치율이 증가함에 따라 계산 시간이 오래 걸릴 뿐 아니라, 최적해를 구하지 못하는 경우도 증가하는 것을 볼 수 있다. 그리고 최대 인덱스가 증가하는 경우도 탐색 공간이 커지므로 동일한 결과를 관찰할 수 있다. 장치율이 70%가 넘어가면, 여덟 단의 장치형태에 대해서는 거의 대부분 해를 찾지 못하는 것도 알 수 있다. 하지만, 일반적인 터미널에서 한 베이가 보통 열 개 이하의 스택과 네 개의 단으로 구성되는 것을 감안하면, 본 연구에서 제안하는 최적화 방법론으로 현장 문제를 해결하는 것이 어느 정도 가능할 것으로 예상된다. 그러나, 여섯 단까지 컨테이너를 적재하는 일부 터미널을 고려한다면, 적용 가능성은 낮아지게 된다.

<Figure 10>은 열 개의 스택과 여섯 단의 베이에서 60%, 70%, 80%의 장치율을 가지는 문제들에 대해 해를 구한 비율과 평균 계산 시간을 나타낸 도표이다. 평균 계산 시간의 경우 성공한 경우만 고려하므로 장치율이 70%와 80%인 경우는 큰 최대 인덱스 P에 대한 결과는 크게 유의미하지 않다. 즉, 최대 인덱스가 15일 때보다 20일 때 더 작은 평균 계산 시간을 보여주지만, 이는 해를 구할 수 있었던 문제들에 대해서만 계산 시간이 고려되었기 때문이다. 모든 컨테이너가 다른 인덱스를 가지는 경우는 실패한 경우를 제외하더라도 계산 시간이 큰 것을 알 수 있다.

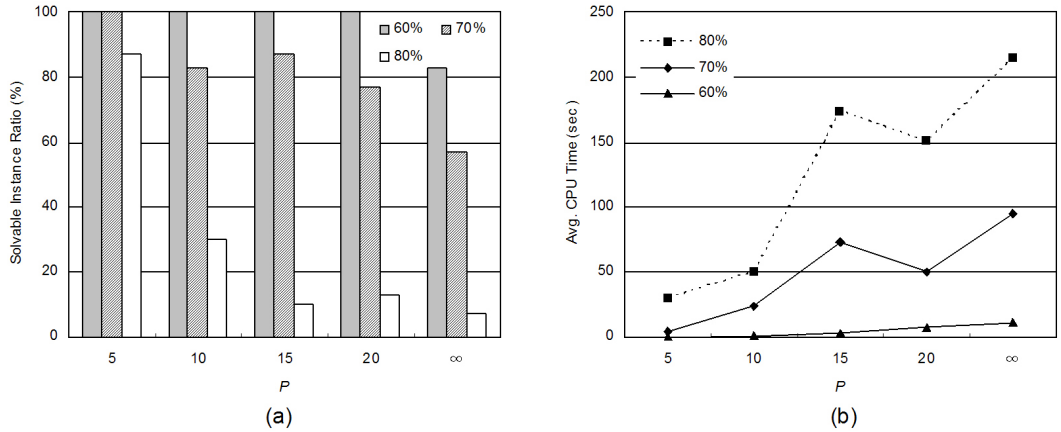

Figure 10. Experimental results of 10 stacks and 6 tiers with different utilization, (a) ratio of problem instances whose optimal solution was found, (b) average CPU time to find the optimal solution

Table 4. Experimental results of random problem instances (CPU Time in sec.)

u (%)	H	W	P = 5				P = 10				P = 15				P = 20				P = ∞			
			Solvable		CPU Time		Solvable		CPU Time		Solvable		CPU Time		Solvable		CPU Time		Solvable		CPU Time	
			S	%	AVG	MAX	S	%	AVG	MAX	S	%	AVG	MAX	S	%	AVG	MAX	S	%	AVG	MAX
40	4	6	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0
		8	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.1	30	100	0.0	0.1	30	100	0.0	0.0
		10	30	100	0.0	0.0	30	100	0.0	0.1	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0
		12	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0
	6	6	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0
		8	30	100	0.0	0.0	30	100	0.0	0.5	30	100	0.0	0.1	30	100	0.0	0.2	30	100	0.0	0.2
		10	30	100	0.0	0.1	30	100	0.0	0.8	30	100	0.0	0.5	30	100	0.5	11.7	30	100	2.2	62.6
		12	30	100	0.0	0.0	30	100	1.3	23.1	30	100	1.1	12.1	30	100	5.5	116.1	30	100	3.8	51.1
	8	6	30	100	0.0	0.1	30	100	0.0	0.4	30	100	0.1	0.9	30	100	0.2	1.1	30	100	0.3	3.2
		8	30	100	0.1	2.1	30	100	0.5	6.4	30	100	3.7	97.6	30	100	7.3	159.2	30	100	5.8	125.4
		10	30	100	0.1	0.4	30	100	6.1	76.3	29	97	1.2	15.8	30	100	5.8	98.4	29	97	3.0	25.2
		12	27	90	6.7	132.3	28	93	2.0	37.2	27	90	6.6	91.4	24	80	6.3	96.3	17	57	24.8	143.1
50	4	6	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0
		8	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0
		10	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.1	30	100	0.0	0.2
		12	30	100	0.0	0.0	30	100	0.0	0.2	30	100	0.0	0.1	30	100	0.2	3.1	30	100	0.0	0.6
	6	6	30	100	0.0	0.1	30	100	0.0	0.2	30	100	0.0	0.3	30	100	0.1	0.5	30	100	0.1	0.4
		8	30	100	0.0	0.3	30	100	0.5	6.9	30	100	0.3	5.5	30	100	0.0	0.3	30	100	0.4	3.1
		10	30	100	0.1	1.5	30	100	1.3	18.8	28	93	1.1	17.9	30	100	0.9	14.7	29	97	4.1	113.4
		12	30	100	0.5	7.9	30	100	2.4	27.1	28	93	1.9	28.6	28	93	6.5	57.8	27	90	1.2	19.4
	8	6	30	100	0.5	5.2	30	100	0.9	9.1	30	100	7.1	123.9	30	100	1.1	6.3	27	90	9.1	78.6
		8	30	100	2.5	33.8	30	100	7.0	64.1	30	100	11.1	88.8	29	97	18.0	135.0	26	87	21.4	134.3
		10	28	93	6.6	113.5	29	97	12.0	135.6	23	77	11.1	119.3	24	80	13.0	137.5	19	63	20.2	133.1
		12	28	93	9.6	89.9	24	80	26.1	142.8	18	60	23.1	119.0	22	73	30.3	166.4	15	50	43.6	151.8
60	4	6	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0	30	100	0.0	0.0
		8	30	100	0.0	0.0	30	100	0.0	0.1	30	100	0.0	0.0	30	100	0.0	0.1	30	100	0.0	0.2
		10	30	100	0.0	0.1	30	100	0.0	0.4	30	100	0.0	0.0	30	100	0.0	0.2	30	100	0.0	0.1
		12	30	100	0.0	0.0	30	100	0.0	0.2	30	100	0.2	5.1	30	100	1.4	38.5	30	100	0.6	10.1
	6	6	30	100	0.1	0.9	30	100	0.4	4.5	30	100	0.6	4.3	30	100	1.5	9.5	30	100	2.4	18.6
		8	30	100	0.2	3.0	30	100	3.5	86.6	30	100	4.5	116.8	30	100	2.1	35.3	30	100	11.2	96.7
		10	30	100	0.2	2.5	30	100	0.8	11.8	30	100	2.7	25.2	30	100	6.9	132.8	25	83	11.8	74.7
		12	30	100	0.9	7.3	29	97	11.3	98.0	26	87	8.4	97.6	25	83	14.9	118.0	22	73	16.3	239.3
	8	6	30	100	21.5	291.3	26	87	26.9	177.6	24	80	49.8	190.6	18	60	46.2	133.2	13	43	68.2	190.7
		8	29	97	12.0	73.2	20	67	63.5	171.4	18	60	34.8	164.5	15	50	91.9	268.4	16	53	63.0	310.3
		10	27	90	13.8	91.3	15	50	66.3	184.1	11	37	68.0	317.2	10	33	47.5	132.7	5	17	138.1	431.2
		12	26	87	51.1	398.6	13	43	57.4	230.3	3	10	36.5	63.2	6	20	29.3	65.8	2	7	326.9	409.4
70	4	6	30	100	0.0	0.1	30	100	0.0	0.1	30	100	0.0	0.1	30	100	0.0	0.2	30	100	0.0	0.3
		8	30	100	0.0	0.1	30	100	0.0	0.1	30	100	0.0	0.7	30	100	0.0	0.1	30	100	0.0	0.2
		10	30	100	0.0	0.1	30	100	0.1	0.5	30	100	0.3	6.3	30	100	0.3	3.9	30	100	1.4	31.6
		12	30	100	0.0	0.3	30	100	0.4	5.1	30	100	3.9	106.0	30	100	2.2	48.0	30	100	3.1	42.1
	6	6	30	100	1.1	18.7	30	100	5.4	54.7	30	100	15.9	273.4	29	97	15.8	120.4	27	90	19.4	191.0
		8	30	100	2.4	22.5	29	97	17.6	99.4	27	90	23.6	118.5	26	87	52.4	203.7	23	77	43.9	225.8
		10	30	100	4.7	74.7	25	83	24.4	150.1	26	87	73.1	442.4	23	77	50.1	186.3	17	57	94.9	405.0
		12	29	97	16.1	206.4	27	90	48.7	370.3	20	67	41.7	214.1	22	73	34.0	146.8	17	57	65.6	556.6
	8	6	27	90	44.5	168.2	9	30	151.8	323.6	3	10	205.1	309.2	0	0	-	-	0	0	-	-
		8	17	57	63.4	210.9	0	0	-	-	3	10	193.1	371.2	1	3	213.0	213.0	0	0	-	-
		10	18	60	78.8	264.0	2	7	189.8	377.4	2	7	65.2	119.5	1	3	75.0	75.0	0	0	-	-
		12	14	47	117.3	495.4	2	7	79.3	109.4	2	7	139.6	209.9	0	0	-	-	0	0	-	-
80	4	6	30	100	0.0	0.1	30	100	0.2	2.6	30	100	0.2	2.7	30	100	0.5	6.5	30	100	1.2	17.8
		8	30	100	0.1	0.9	30	100	0.2	1.2	30	100	0.3	1.4	30	100	0.5	6.3	30	100	2.7	56.4
		10	30	100	0.1	1.2	30	100	0.5	3.8	30	100	2.2	24.2	30	100	3.0	45.6	30	100	6.3	117.7
		12	30	100	0.3	3.7	30	100	1.6	15.4	30	100	1.7	16.8	30	100	3.6	28.2	30	100	10.1	244.6
	6	6	29	97	25.5	186.0	18	60	65.0	259.5	11	37	71.4	195.7	10	33	74.2	230.6	4	13	105.5	219.2
		8	29	97	26.0	302.9	17	57	76.0	318.5	13	43	75.0	272.7	12	40	110.5	256.3	5	17	159.9	326.0
		10	26	87	30.4	144.3	9	30	49.9	184.4	3	10	173.6	436.5	4	13	150.3	468.8	2	7	214.2	246.4
		12	26	87	96.3	364.1	6	20	136.7	251.5	1	3	176.8	176.8	3	10	357.6	859.2	1	3	359.2	359.2
	8	6	1	3	55.5	55.5	0	0	-	-	0	0	-	-	0	0	-	-	0	0	-	-
		8	1	3	459.6	459.6	0	0	-	-	0	0	-	-	0	0	-	-	0	0	-	-
		10	4	13	153.0	260.1	0	0	-	-	0	0	-	-	0	0	-	-	0	0	-	-
		12	2	7	84.0	112.4	0	0	-	-	0	0	-	-	0	0	-	-	0	0	-	-

<Figure 11>은 계산 시간의 전형적인 분포를 보여준다. 열 개의 스택과 네 단으로 이루어진 베이에서 80%의 장치율과 최대 인덱스 10의 설정을 사용한 결과이며, 30개의 모든 문제에 대해 최적해를 도출한 경우이다. 대부분이 1초 이내의 시간에 해결되었으나, 몇몇 문제를 해결하기 위해서는 상대적 으로 큰 시간이 필요한 것을 알 수 있다. 이는, 비록 문제를 생성하기 위해 사용된 설정이 같더라도, 서로 다른 인덱스를 가진 컨테이너의 분포에 따라 탐색 공간의 크기와 탐색의 효율성이 달라지기 때문으로 파악된다.

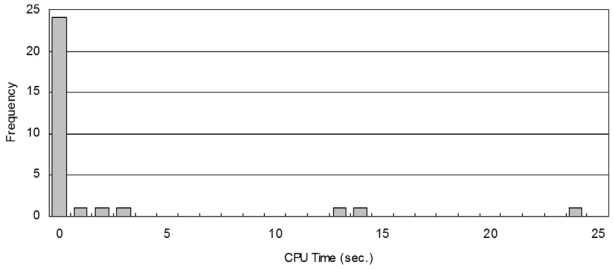

Figure 11. Distribution of the CPU time forsolving instances with 10 stacks, 4 tiers, 80% utilization, and P = 10

6. 결 론

본 연구에서는 컨테이너 터미널의 생산성을 향상시키기 위한 방법 중 하나인 베이 내 컨테이너 재정돈을 위한 최적 탐색 방안을 제시하였다. 이를 위해 그래프 상에서 최단 경로를 탐색하는 문제로 형식화하고 A* 알고리즘을 개발하여 최적 이적 순서를 수립하는 방법을 개발하였다. 재정돈 문제를 위한 A* 알고리즘에서의 휴리스틱 추정 값을 구하기 위해 컨테이너 재배치 문제를 도입하였으며 역시 A* 알고리즘을 사용하여 최적해를 도출하였다.

제시한 알고리즘의 성능은 수치 실험을 통해 검증하였다. 먼저, 기존 연구 결과와 비교하여 최적해를 보다 짧은 시간에 구할 수 있음을 보였다. 다음으로, 임의로 생성한 문제를 사용하여 알고리즘의 전반적인 성능을 검토하였다. 실험 결과, 실용적인 크기의 문제에 대해서는 본 연구의 방법론이 어느 정도 효과적임을 알 수 있었으나, 큰 문제의 경우 추가적인 연구가 필요함을 확인할 수 있었다.

A* 알고리즘을 사용한 접근 방법의 한계는 분명하다. 먼저, 주어진 문제에 대해 충분한 시간을 투자하기 전에는 해의 도출 가능성을 미리 알기 힘들다. 또한 작은 크기의 문제에 대해서도 저장 공간의 부족으로 해를 찾지 못할 가능성이 존재하므로 현장에 적용하는 데 어려움을 가진다. 그리고, 최적해를 찾기 전에는 어떠한 가능해도 도출되지 않는다. 따라서 실제 문제의 해결에 적용하기 위해서는 탐욕 전략(greedy strategy) 등을 사용하여 단계적으로 향상된 해를 제시할 수 있는 온라인(online) 접근 방법이 필요할 것이다.

큰 문제에 대하여 안정적으로 최적해를 도출하기 위한 추후 연구도 필요하다. 이를 위해 A* 알고리즘의 저장 공간 부족 문제를 해결할 수 있는 재귀적 최상우선 탐색(recursive best-first search) 등을 사용하는 방법을 고려할 수 있다(Russell and Norvig, 2003). 하지만, 동일한 노드의 재방문이 빈번하게 이루어지는 문제의 특성 상 그와 같은 방법은 계산 시간의 막대한 증가를 피하지 못할 것으로 예상된다. 따라서 현재 휴리스틱 추정 값으로 사용하는 최소 재배치 컨테이너 개수보다 더 개선된 추정 방법의 개발이 보다 효과적인 방안이 될 것으로 보인다. 또한 A* 알고리즘 아닌 다른 최적화 방법론을 도입하여 문제를 해결하는 것도 가능할 것이다. 예를 들어, 해 공간을 효과적으로 구분할 수 있는 분지(branching) 기법을 개발한다면 분지한계법(branch and bound)의 적용도 유용할 것으로 생각된다.

본 연구에서 제시된 결과를 터미널 현장에 보다 광범위하게 적용하기 위해서는 다음과 같은 추가적인 요구 사항들도 고려해야 할 것이다. 먼저, 터미널에서는 재정돈 시 적하를 위한 반출 순서가 명확하게 정해지지 않고 불확실성이 존재하며, 운영 방식에 따라 주어진 순서와 어느 정도 다르게 반출하는 것도 가능하다. 또한, 컨테이너가 실릴 선박, 목적항(port of discharge), 무게 등을 사용하여 상대적인 반출 순서를 유추해야 하는 상황도 일반적이다. 다음으로, 컨테이너 취급 작업의 편이성과 추가적이 컨테이너 반입 등을 고려하는 경우, 동일한 최종형태라고 하더라도 실제 선호는 달라질 수 있다. 마지막으로, 장치율이 높은 경우 최종형태로 만들 수 없는 경우도 존재하며, 항상 많은 이적 작업을 통해 최종형태로 만드는 것만이 최선은 아니다. 따라서, 이와 같은 다양한 요구 사항을 반영하여 현장에 적용할 수 있도록 하는 절차의 개발이 필요하다.

<부 록>

정리 2의 증명을 위해, $j = 1, \cdots, W$와 $k = 1, \cdots, n_j$에 대하여, 주어진 장치형태의 특정 슬롯에 인덱스 i인 적합 컨테이너가 위치한 여부를 나타내는 $\delta_j^i(k)$를 도입하자. 즉,

$$\delta_j^i(k) = \begin{cases} 1 & \text{if a container with index } i \text{ is} \\ & \text{located at tier } k \text{ of stack } j \\ 0 & \text{otherwise} \end{cases} \quad (14)$$

일반화를 위해, $k = 0$인 경우, $i = P$일 때 $\delta_j^i(k)$는 1을 가지고, 그렇지 않은 경우 0을 가진다고 정의한다. 한 슬롯에는 하나의 컨테이너만 장치되므로, $j = 1, \cdots, W$와 $k = 0, \cdots, n_j$에 대하여, $\sum_{i=1}^{P} \delta_j^i(k) = 1$이다.

스택 j에서 q개의 적합 컨테이너가 재배치된다고 하자. 이 때, 스택에서 재배치되지 않고 남아있는 최상단의 컨테이너는

단 (n_j-q)에 위치하며, 그 컨테이너에 의해 가능 슬롯의 인덱스가 결정된다. 그리고, $r(i, j)$개 이상의 적합 컨테이너가 재배치될 때만 인덱스 i의 재배치 컨테이너가 최종형태에서 그 스택에 장치 가능하므로, 즉, 가능 슬롯의 인덱스가 i 이상이 되므로, 이는 곧 최상단 컨테이너의 인덱스가 i 이상인 것을 의미한다. 따라서, 아래가 성립한다.

$$\sum_{l=1}^{P}\delta_j^l(n_j-q)=\begin{cases}1, & \text{if } q\ge r(i,j)\\ 0, & \text{otherwise}\end{cases}\quad \text{for } i=1,\cdots,P \tag{15}$$

추가로, 스택 j에서 x_j개의 적합 컨테이너를 재배치할 때, 그 스택에 있는 인덱스 i를 가지는 가능 슬롯의 개수 $a_j^i(x_j)$와, 그 스택에서 재배치되는 인덱스 i의 적합 컨테이너 개수 $b_j^i(x_j)$를 도입하자. 이 경우, $(H-n_j+x_j)$는 가능 슬롯의 개수를 의미하므로, 이들은 δ_j^i를 사용하면 다음과 같이 쓸 수 있다.

$$a_j^i(x_j)=(H-n_j+x_j)\delta_j^i(n_j-x_j) \tag{16}$$

$$b_j^i(x_j)=\sum_{q=0}^{x_j-1}\delta_j^i(n_j-q) \tag{17}$$

정리 2의 증명 : j = 1, …, W에 대하여 재배치 계획 x_j 주어졌을 때, i = 1, …, P에 대하여, 보조정리 1의 식 (1)은 a_j^i와 b_j^i, c^i를 사용하여 다음과 같이 다시 쓸 수 있다:

$$\sum_{l=i}^{P}\sum_{j=1}^{W}a_j^l(x_j)\ge\sum_{l=i}^{P}\sum_{j=1}^{W}b_j^l(x_j)+\sum_{l=i}^{P}c^l \tag{18}$$

즉,

$$\sum_{j=1}^{W}\left(\sum_{l=i}^{P}a_j^l(x_j)-\sum_{l=i}^{P}b_j^l(x_j)\right)\ge\sum_{l=i}^{P}c^l \tag{19}$$

식 (19)의 좌변에서 합계의 내부 항을 고려하자. 먼저, $x_j\ge r(i, j)$인 경우, 식 (16)과 식 (17)을 적용하면, 식 (15)에 의하여,

$$\sum_{l=i}^{P}a_j^l(x_j)-\sum_{l=i}^{P}b_j^l(x_j)$$

$$=\sum_{l=i}^{P}(H-n_j+x_j)\delta_j^l(n_j-x_j)-\sum_{l=i}^{P}\sum_{q=0}^{x_j-1}\delta_j^l(n_j-q) \tag{20}$$

$$=(H-n_j+x_j)\sum_{l=i}^{P}\delta_j^l(n_j-x_j)-\sum_{q=0}^{r(i,j)-1}\sum_{l=i}^{P}\delta_j^l(n_j-q) \tag{21}$$

$$-\sum_{q=r(i,j)}^{x_j-1}\sum_{l=i}^{P}\delta_j^l(n_j-q)$$

$$=(H-n_j+x_j)-(x_j-r(i,j))=H-n_j+r(i,j) \tag{22}$$

다음으로, $x_j<r(i,j)$인 경우, 역시 식 (15)에 의하여,

$$\sum_{l=i}^{P}a_j^l(x_j)-\sum_{l=i}^{P}b_j^l(x_j)=(H-n_j+x_j)\sum_{l=i}^{P}\delta_j^l(n_j-x_j) \tag{23}$$

$$-\sum_{q=0}^{x_j-1}\sum_{l=i}^{P}\delta_j^l(n_j-q)=0$$

식 (3)에서 주어진 y_{jk}의 정의에 의해, 식 (22)와 식 (23)은 다음과 같이 통합된다 :

$$\sum_{l=i}^{P}a_j^l(x_j)-\sum_{l=i}^{P}b_j^l(x_j)=(H-n_j+r(i,j))y_{ir(i,j)} \tag{24}$$

그러므로, 최종형태가 가능한 필요충분조건은 식 (4)와 같다. □

참고문헌

Caserta, M., Voβ, S., and Sniedovich, M. (2011), Applying the Corridor Method to a Blocks Relocation Problem, *OR Spectrum*, **33**, 915-929.

Caserta, M. and Voβ, S. (2009), A Corridor Method-Based Algorithm for the Pre-marshalling Problem, *Lecture Notes in Computer Science*, **5484**, 788-797.

Choe, R., Park, T., Oh, M.-S., Kang, J., and Ryu, K. R. (2011), Generating a Rehandling-free Intra-block Remarshaling Plan for an Automated Container Yard, *Journal of Intelligent Manufacturing*, **22**, 201-217.

Huang, S.-H. and Lin, T.-H. (2012), Heuristic Algorithms for Container Pre-marshalling Problems, *Computers and Industrial Engineering*, **62**(1), 13-20.

Imai, A., Sasaki, K., Nishimura, E., and Papadimitriou, S. (2006), Multi-objective Simultaneous Stowage and Load Planning for a Container Ship with Container Rehandle in Yard Stacks, *European Journal of Operational Research*, **171**(2), 373-389.

Jeong, Y. H., Kim, K. H., Woo, Y. J., and Seo, B. H. (2012), A Simulation Study on a Workload-based Operation Planning Method in Container Terminals, *Industrial Engineering and Management Systems*, **11**(1), 103-113.

Kang, J., Ryu, K. R., and Kim, K. H. (2006), Deriving Stacking Strategies for Export Containers with Uncertain Weight Information, *Journal of Intelligent Manufacturing*, **17**, 399-410.

Kim, K. H. (2007), Decision-making Problems for the Operation of Container Terminals, *Journal of the Korean Institute of Industrial Engineers*, **33**(3), 290-302.

Kim, K. H. and Bae, J. W. (1998), Re-marshalling Export Containers in Port Container Terminals, *Computers and Industrial Engineering*, **35**(3-4), 655-658.

Kim, K. H. and Hong, G.-P. (2006), A Heuristic Rule for Relocating Blocks, *Computers and Operations Research*, **33**, 940-954.

Kim, K. H., Kang, J. S., and Ryu, K. R. (2004), A Beam Search Algorithm for the Load Sequencing of Outbound Containers in Port Container Terminals, *OR Spectrum*, **26**(1), 93-116.

Kim, K. T. and Kim, K. M. (2011), Metaheuristics of the Rail Crane Scheduling Problem, *IE Interface*, **24**(4), 281-294.

Kim, K. H., Kim, K. Y., and Ko, C. S. (1997), Load Scheduling Using a Genetic Algorithm in Port Container Terminals, *Journal of the Korean Institute of Industrial Engineers*, **23**(4), 645-660.

Kim, K. H. and Park, Y. M. (1996), A Slot Assignment Method in the Container Yard for Export Containers Considering Their Weights, *Journal of the Korean Institute of Industrial Engineers*, **22**(4), 753-770.

Kim, K. H. and Park, K. T. (2003), A Note on a Dynamic Space-allocation Method for Outbound Containers, *European Journal of*

Operational Research, **148**, 92-101.

Kim, K. H., Park, Y. M., and Ryu, K.-R. (2000), Deriving Decision Rules to Locate Export Containers in Container Yards, *European Journal of Operational Research*, **124**, 89-101.

Ku, L. P., Lee, L. H., Chew, E. P., and Tan, K. C. (2010), An Optimisation Framework for Yard Planning in a Container Terminal : Case with Automated Rail-mounted Gantry Cranes, *OR Spectrum*, **32**, 519-541.

Lee, S. W. (2011), A Genetic Algorithm for the Container Pick-Up Problem, *IE Interface*, **24**(4), 362-372.

Lee, Y. and Chao, S-L. (2009), A Neighborhood Search Heuristic for Pre-marshalling Export Containers, *European Journal of Operational Research*, **196**, 468-475.

Lee, Y. and Lee Y.-J. (2010), A Heuristic for Retrieving Containers from a Yard, *Computers and Operations Research*, **37**, 1139-1147.

Lee, Y. and Hsu, N.-Y. (2007), An Optimization Model for the Container Pre-marshalling Problem, *Computers and Operations Research*, **34**, 3295-3313.

Nilsson, N. J. (1998), *Artificial Intelligence: A New Synthesis*, Morgan Kaufmann Publishers, Inc., San Francisco, CA.

Russell, S. and Norvig, P. (2003), *Artificial Intelligence : A Modern Approach*, 2nd Ed., Pearson Education, Upper Saddle River, NJ.

Wan, Y.-W., Liu, J., and Tsai, P.-C. (2009), The Assignment of Storage Locations to Containers for a Container Stack, *Naval Research Logistics*, **56**, 699-713.

Winston, W. L. (2004), *Operations Research : Applications and Algorithms*, 4th Ed., Thomson Learning, Inc., Belmont, CA.

Won, S. H. and Kim, K. H. (2009), Yard Planning Considering the Load Profile of Resources in Container Terminals, *Journal of the Korean Institute of Industrial Engineers*, **35**(1), 58-72.

Zhang, C., Liu, J., Wan, Y-W., Murty, K. G., and Linn, R. J. (2003), Storage Space Allocation in Container Terminals, *Transportation Research Part B*, **37**, 883-903.