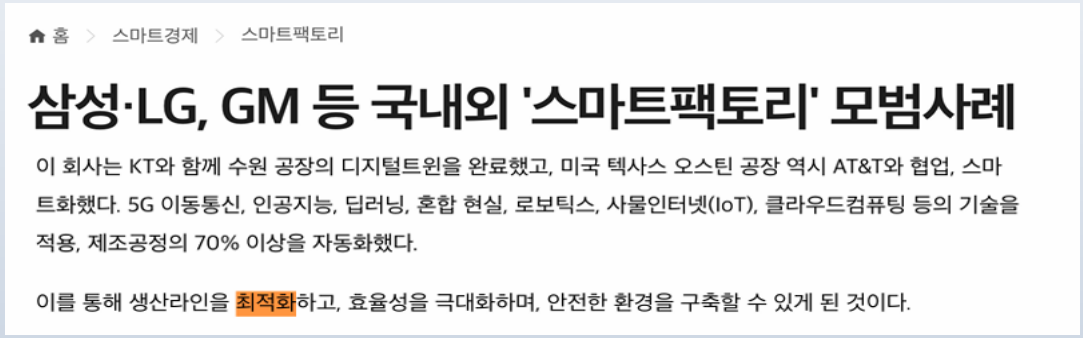
산업 공학을 기반한 중소기업 컨설팅; 작업설계 원리를 기반한 최적화를 중심으로

2024-1 파란학기 맥킨지 아주

팀장: 김재연
팀원: 김태훈, 신수연, 정호준

01 주제 선정 배경


홈 > 스마트경제 > 스마트팩토리

삼성·LG, GM 등 국내외 '스마트팩토리' 모범사례

이 회사는 KT와 함께 수원 공장의 디지털트윈을 완료했고, 미국 텍사스 오스틴 공장 역시 AT&T와 협업, 스마트화했다. 5G 이동통신, 인공지능, 딥러닝, 혼합 현실, 로보틱스, 사물인터넷(IoT), 클라우드컴퓨팅 등의 기술을 적용, 제조공정의 70% 이상을 자동화했다.

이를 통해 생산라인을 최적화하고, 효율성을 극대화하며, 안전한 환경을 구축할 수 있게 된 것이다.

[대기업의 최적화 적용 사례]

SAMSUNG 삼성SDS LG CNS

IBM

[IT 컨설팅 기업]

대기업은 전체 최적화를 위한 컨설팅을 받고 있음

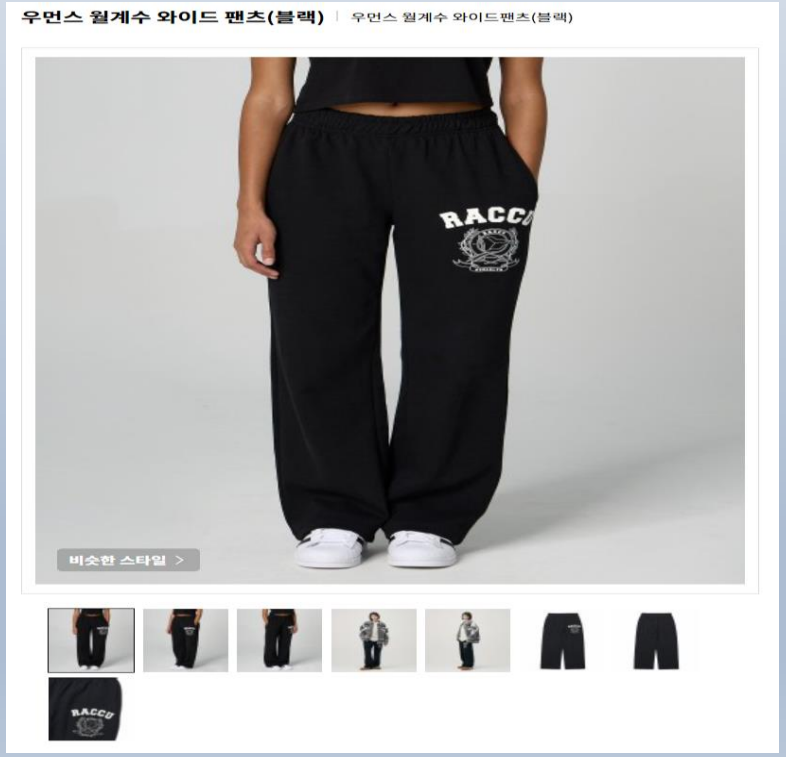
↓

중소기업은 다양한 측면에서 최적화를 고려하지 못하는 경우가 많음

02 대상 기업, 모델 선정

TAKEASY

패션 브랜드 ‘TAKEASY’의 인기 상품인
‘월계수 와이드 팬츠’를 대상 제품으로 선정

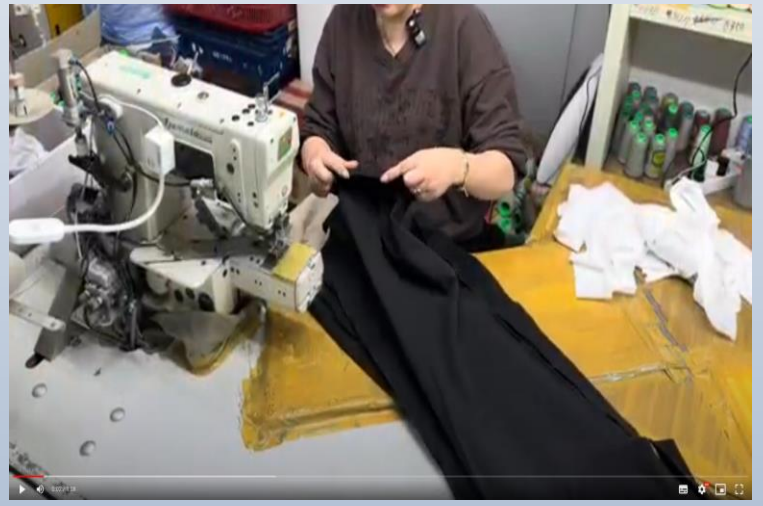


‘월계수 와이드 팬츠’ 의 생산 리드타임 단축,
의류 생산 작업자들의 작업 수정을 통해
생산성 증가를 목표로 설정

03 의류 공장 방문

- 브랜드가 디자인을 하고, 의류는 아웃소싱을 통해 생산하는 방식
- 의류 생산 공장을 방문해 작업영상을 촬영하고, 최적화에 필요한 데이터를 수집
- 의류 제작 과정을 파악

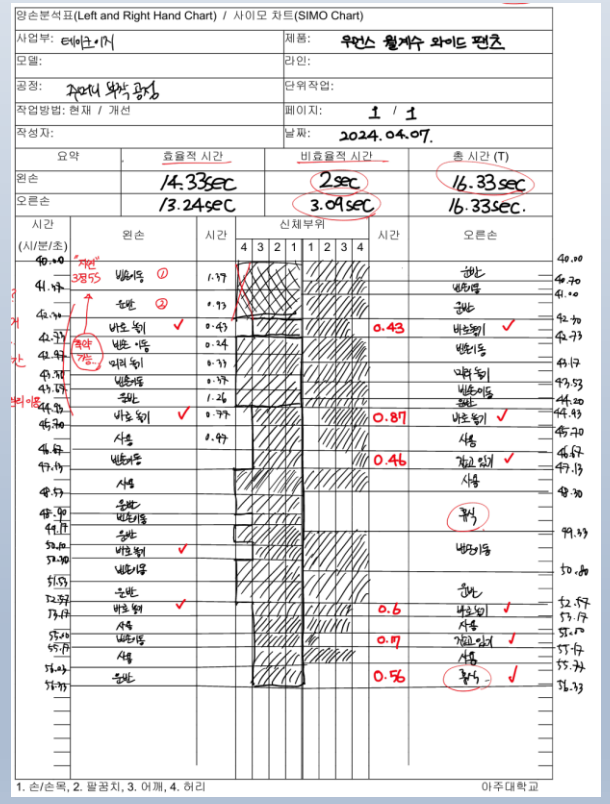
2023년 월계수 판매(월별).xlsx
유효기간 2024-05-28
36.77KB



[단위 작업 별 영상 촬영 및 수요 데이터 수집]

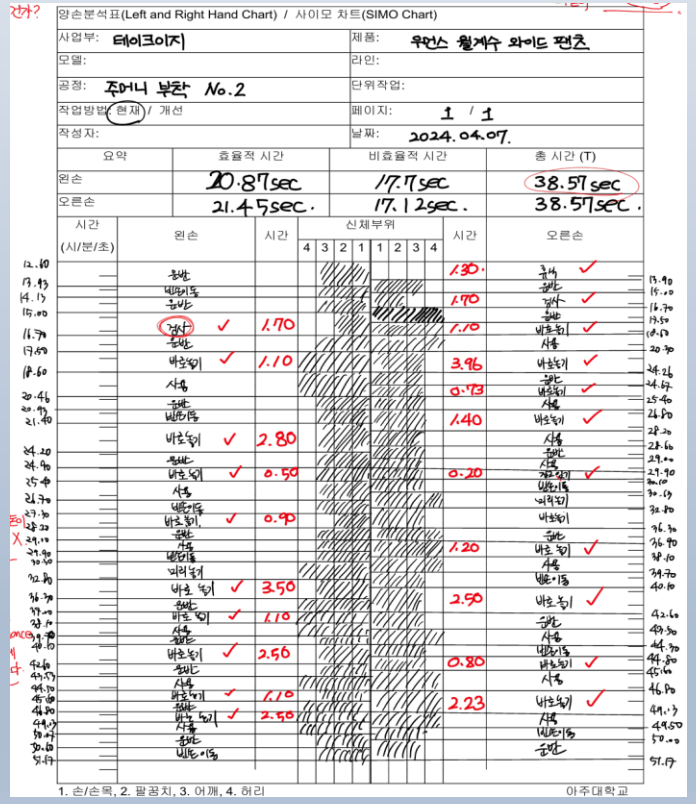
04 작업 개선(양손분석표 작성)

- 양손분석표 작성을 통해 각 작업에서 이루어지고 있는 세부 동작을 파악
- 비효율적인 서어블릭을 찾아내고, 개선점을 제안


양손분석표(Left and Right Hand Chart) / 사이모 차트(SIMO Chart)

사업부: 테이크이지 | 제품: 우먼스 월계수 와이드 팬츠
모델: | 라인:
공정: 주머니 봉작 공정 | 단위작업:
작업방법: 현재 / 개선 | 페이지: 1 / 1
작성자: | 날짜: 2024.04.07.

요약	효율적 시간	비효율적 시간	총 시간 (T)
왼손	14.33sec	2sec	16.33sec
오른손	13.24sec	3.09sec	16.33sec.


양손분석표(Left and Right Hand Chart) / 사이모 차트(SIMO Chart)

사업부: 테이크이지 | 제품: 우먼스 월계수 와이드 팬츠
모델: | 라인:
공정: 주머니 부착 No.2 | 단위작업:
작업방법: (현재) / 개선 | 페이지: 1 / 1
작성자: | 날짜: 2024.04.07.

요약	효율적 시간	비효율적 시간	총 시간 (T)
왼손	20.87sec	17.71sec	38.57sec
오른손	21.45sec.	17.12sec.	38.57sec.


양손분석표(Left and Right Hand Chart) / 사이모 차트(SIMO Chart)

사업부: 테이크이지 | 제품: 우먼스 월계수 와이드 팬츠
모델: | 라인:
공정: 겉메선 봉제 | 단위작업:
작업방법: (현재) / 개선 | 페이지: 1 / 1
작성자: | 날짜: 2024.04.07.

요약	효율적 시간	비효율적 시간	총 시간 (T)
왼손	20.11sec	23.49sec	43.60sec.
오른손	20.68sec.	22.92sec.	43.60sec.

[양손분석표 작성 결과]

04 작업 개선(문제점 및 해결책 파악)

1. 작업자는 앉아서 작업 → 허리 사용 많음

『회전의자를 두어, 작업자가 허리를 움직이지 않고도 작업반경을 늘릴 수 있도록 개선안을 제시』
『작업 시 사용되는 신체부위를 최소화할 수 있도록 작업방식을 수정(어깨,손목까지 신체를 사용하도록 유도)』

2. 작업시간이 길어지는 주된 요인은 '잡기' 동작

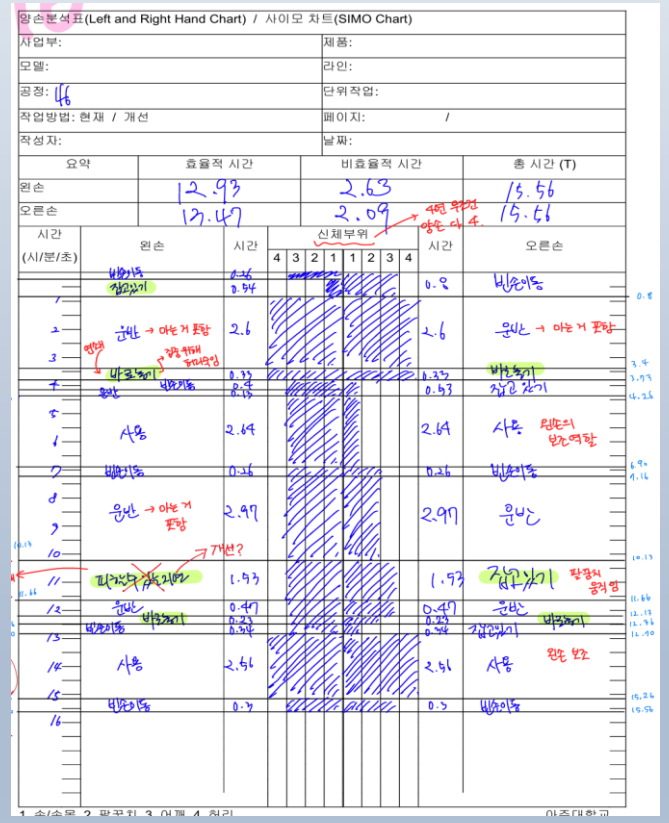
- '잡기' 동작이 발생하는 유형
1. 라벨을 부착해야 하는 작업에서 라벨이 손에 잘 잡히지 않는 경우
2. 다림질하는 작업에서 먼지를 떼어내려 하는 경우가 발생

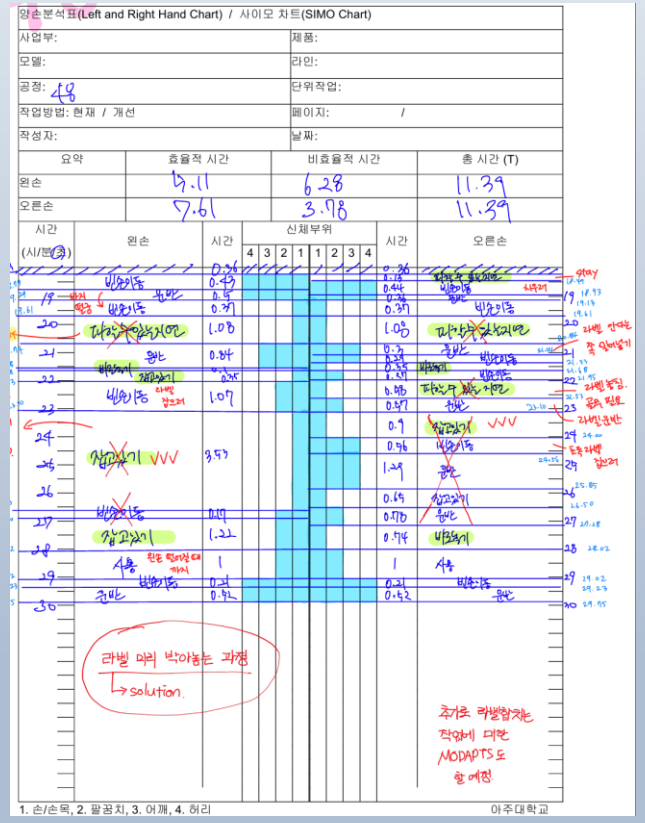
『이러한 점에서는 골무를 착용할 것을 제안하여, 비효율적인 시간이 사용되는 것을 방지』

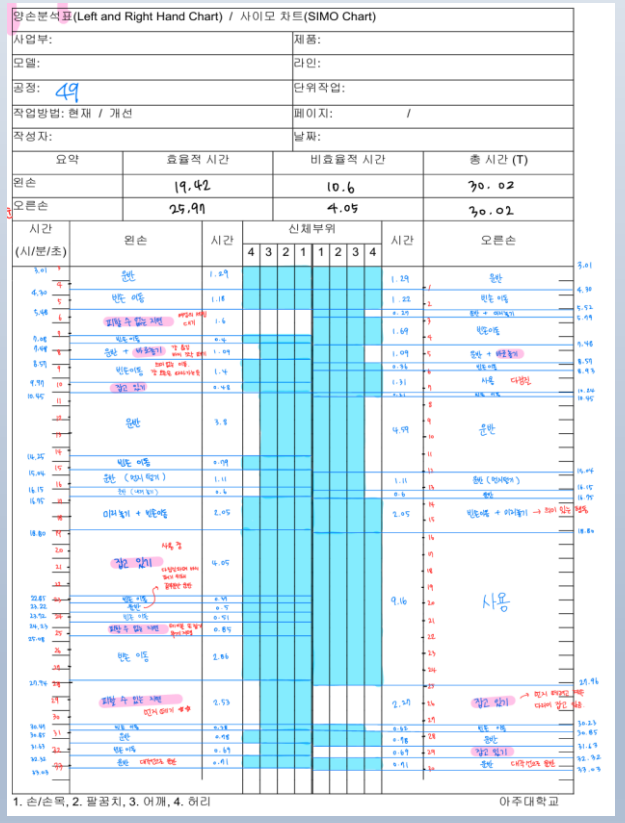
개선점을 적용하였다고 가정하고, MODAPTS를 작성하며 표준시간을 산정

04 작업 개선(MODAPTS 표 작성)

• 개선안을 기반으로 MODAPTS 원리에 따라 작업의 표준시간 예측
(신체 사용 정도, 동작의 종류에 따라 MOD 단위로 시간이 정량적으로 제시된 원칙)







[MODAPTS 작성 결과]

04 작업 개선(작업 별 표준 시간 정리)

	양손분석표	MODAPTS	개선(%)
1번 작업: 주머니 옆면 재봉작업_시작	16.33	13.30	-18.55
2번 작업: 주머니 옆면 재봉 작업_마무리	38.57	15.98	-58.57
3번 작업: 바지 앞,뒷면 결합_재봉 작업	43.60	28.04	-35.69
4번 작업: 바지 밑단 재봉 작업	15.56	7.86	-49.51
5번 작업: 고무줄 재봉 공정	12.32	12.05	-2.19
6번 작업: 라벨 재봉 작업	11.39	10.43	-8.43
7번 작업: 남녀 다림질 작업	30.02	17.34	-42.25

- 작업은 총 7개의 단위 공정으로 진행됨
- 모든 작업에서 표준시간의 개선이 이루어짐
- 비효율적인 요소가 많았던 작업에서는 최소 **30%** 대의 표준시간 개선이 이루어짐
- 앞서 언급했던 문제점을 기반으로 지연시간을 모두 제거한 결과 표준시간 측면에서 상당한 개선을 보임
- 표준시간 개선은 생산성 개선에도 영향이 많음

05 최적화 Line Balancing & Makespan 최소화 검증

해당 공정은 모든 작업이 flow shop 형태
(flow shop: 여러 기계나 작업장을 순서대로 배치해 놓은 상태, 대량 생산에 용이)

Makespan: 작업을 완료하는 데 걸리는 시간
→ flow shop 형태의 공정은 병목 현상의 원인이 되는, 작업 시간이 가장 긴 작업에 종속됨

Objective: batch 크기와 병목 현상의 원인이 되는 작업을 앞/뒤 작업이 도와주는 양을 조절해 Makespan을 최소화

05 최적화 Line Balancing & Makespan 최소화 검증

목적함수: “Makespan”을 최소화

$$\text{minimize Makespan} = \left\lceil \frac{O}{b} \right\rceil * \max(q_{ijk}) + \sum_i \sum_j \sum_k q_{ijk}$$

$$q_{ijk} = \begin{cases} ct_i * b + \text{moving time} + \text{setup time} \ (k = 0) \\ ct_i * b + ct_j * a_{ij} + \text{moving time} + \text{setup time} \ (k = 1) \\ ct_i * (b - a_{ij}) + \text{moving time} + \text{setup time} \ (k = 2) \end{cases}$$

q_{ijk} : production time per batch

ct_i : Cycle time of work i

O : Order quantity,

b : batch size

index) $i = \{1 \dots 7\}$, work number

$j = \{1 \dots 7\}$, work number to help or receive

$k = 0(\text{NONE}), 1(\text{help}), 2(\text{receive})$

목적함수와 Index, 주요 Parameter 설명]

05 최적화 Line Balancing & Makespan 최소화 검증

Tabu-Search Algorithm으로 최적해 도출

```
import random
import math

# 파라미터 정의
ct = [13.3, 15.98, 28.04, 7.86, 12.05, 10.43, 17.34]
st = 5
st_ = 8
mt = 15
oq = 1000

# 변경할 위치 정의
positions = [(0, 1), (1, 0), (1, 2), (2, 1), (2, 3), (3, 2), (3, 4), (4, 3), (4, 5), (5, 4)]
positions.reverse()

# 초기 a 행렬
a_initial = [[0]*7 for _ in range(7)]
```

- **파라미터 설정**(cycle time, setup time 등)
- **인접 작업 간에 도움을 주고 받는다는 조건 설정**

```
[ ]  # q 정의 (k=1 : 도움 주는 것, k=2: 도움 받는 것)
     def q(i, j, b, a):
         if i == 6 or i == j:
             return b * ct[i] + mt + st

         if a[i][j] == 0:  # a[i][j]가 0일 때
             return b * ct[i] + mt + st
         elif a[i][j] > 0:  # 도움 주는 것
             return b * ct[i] + a[i][j] * ct[j] + mt + st_
         else:  # 도움 받는 것
             return (b + a[i][j]) * ct[i] + mt + st

     # 제약 조건
     def constraint(a):
         for i in range(7):
             for j in range(7):
                 if abs(i - j) > 1:
                     a[i][j] = 0
                 a[j][i] = -a[i][j]
                 if i == j:
                     a[i][j] == 0
             for j in range(1,6):
                 if a[i][j-1] > 0:
                     a[i][j+1] <= 0
         return a
```

05 최적화 Line Balancing & Makespan 최소화 검증

Makespan 계산 코드

```
[ ]  # makespan 계산 함수
     def calculate_makespan(b, a):
         q_array = []
         q_list = []
         for i in range(7):
             q_array.append([q(i, j, b, a) for j in range(7)])
         for i in range(7):
             flag = True
             for j in range(7):
                 if a[i][j] > 0:
                     q_list.append(q_array[i][j])
                     flag = False
                 elif a[i][j] < 0:
                     q_list.append(q_array[i][j])
                     flag = False
             if flag:
                 q_list.append(q_array[i][0])

         return max(q_list) * math.ceil(oq / b) + sum(q_list)
```

Tabu-Search 알고리즘 구현

```
[ ]  # 타부 탐색 알고리즘
     def tabu_search(iterations, tabu_tenure, neighborhood_size):
         def get_neighbors(a):
             neighbors = []
             for _ in range(neighborhood_size):
                 neighbor = [row[:] for row in a]
                 i, j = random.choice(positions)
                 neighbor[i][j] += int(random.choice([-5, 5]))
                 neighbors.append(constraint(neighbor))
             return neighbors

         a_current = constraint(a_initial)
         b_current = 1
         min_makespan = calculate_makespan(b_current, a_current)

         best_a = a_current
         best_b = b_current
         best_makespan = min_makespan

         tabu_list = []

         for iteration in range(iterations):
             neighbors = get_neighbors(a_current)
             best_neighbor = None
             best_neighbor_makespan = float('inf')

             for neighbor in neighbors:
                 for b in range(1, 51):
                     neighbor_makespan = calculate_makespan(b, neighbor)
                     if neighbor_makespan < best_neighbor_makespan and neighbor not in tabu_list:
                         best_neighbor = neighbor
                         best_neighbor_makespan = neighbor_makespan
                         best_b_neighbor = b

             if best_neighbor is not None:
                 a_current = constraint(best_neighbor)
                 b_current = best_b_neighbor
                 min_makespan = best_neighbor_makespan

                 if min_makespan < best_makespan:
                     best_a = constraint(a_current)
                     best_b = b_current
                     best_makespan = min_makespan

                 tabu_list.append(a_current)
                 if len(tabu_list) > tabu_tenure:
                     tabu_list.pop(0)

             # 디버깅 로그 추가
             print(f"Iteration {iteration}: Current Best Makespan = {best_makespan}, b = {best_b}")

         return best_a, best_b, best_makespan
```

05 최적화 Line Balancing & Makespan 최소화 검증

Iteration 만큼 실행

```
[ ]  # 타부 탐색 실행
     iterations = 1000
     tabu_tenure = 50
     neighborhood_size = 20

     optimal_a, optimal_b, optimal_makespan = tabu_search(iterations, tabu_tenure, neighborhood_size)
```

```
Iteration 0: Current Best Makespan = 21286.28, b = 14
Iteration 1: Current Best Makespan = 21286.28, b = 14
Iteration 2: Current Best Makespan = 21286.28, b = 14
Iteration 3: Current Best Makespan = 21286.28, b = 14
Iteration 4: Current Best Makespan = 21286.28, b = 14
Iteration 5: Current Best Makespan = 21286.28, b = 14
Iteration 6: Current Best Makespan = 21286.28, b = 14
Iteration 7: Current Best Makespan = 21286.28, b = 14
Iteration 8: Current Best Makespan = 21286.28, b = 14
Iteration 9: Current Best Makespan = 21286.28, b = 14
Iteration 10: Current Best Makespan = 21286.28, b = 14
Iteration 11: Current Best Makespan = 21286.28, b = 14
Iteration 12: Current Best Makespan = 21286.28, b = 14
Iteration 13: Current Best Makespan = 21286.28, b = 14
Iteration 14: Current Best Makespan = 21286.28, b = 14
Iteration 15: Current Best Makespan = 21286.28, b = 14
Iteration 16: Current Best Makespan = 21286.28, b = 14
Iteration 17: Current Best Makespan = 21286.28, b = 14
Iteration 18: Current Best Makespan = 21286.28, b = 14
Iteration 19: Current Best Makespan = 21286.28, b = 14
Iteration 20: Current Best Makespan = 21286.28, b = 14
Iteration 21: Current Best Makespan = 21286.28, b = 14
Iteration 22: Current Best Makespan = 21286.28, b = 14
Iteration 23: Current Best Makespan = 21286.28, b = 14
Iteration 24: Current Best Makespan = 21286.28, b = 14
```

최적 Makespan 결과 출력

```
[ ]  # 결과 출력
     print("a : ")
     for row in optimal_a:
         print(row)
     print("b :", optimal_b)
     print("최적 makespan :", optimal_makespan)
```

생산 시간 확인을 통해 Line Balancing 여부 파악

```
[ ]  q_array = []
     q_list = []
     for i in range(7):
         q_array.append([q(i, j, optimal_b, optimal_a) for j in range(7)])
     for i in range(7):
         flag = True
         for j in range(7):
             if optimal_a[i][j] > 0:
                 q_list.append(q_array[i][j])
                 flag = False
             elif optimal_a[i][j] < 0:
                 q_list.append(q_array[i][j])
                 flag = False
         if flag:
             q_list.append(q_array[i][0])

     print(q_list)
```

05 최적화 Line Balancing & Makespan 최소화 검증

결과 확인

```
a :
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, -5, 0, 0, 0]
[0, 0, 5, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
b : 14
최적 makespan : 21286.28
```

- 3번 작업이 4번 작업으로부터 batch 당 5개씩 도움을 받음 (사이클타임 때문)
- batch size: 14
- 최적 makespan: 21286초(5.91시간)

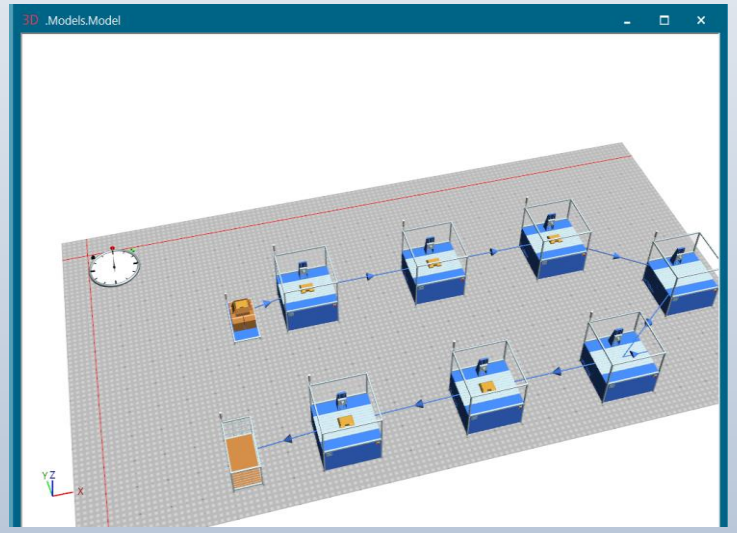
각 작업 별 생산시간 파악

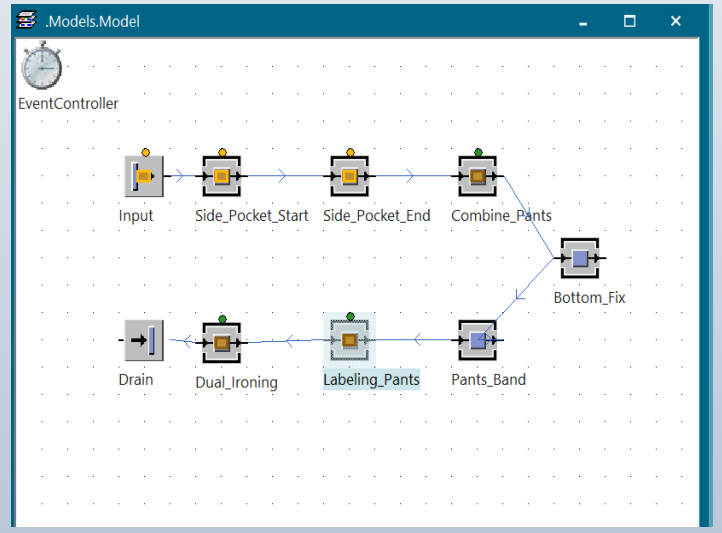
작업 번호	Production time (per batch)
1	206.20 sec
2	243.72 sec
3	272.36 sec
4	273.24 sec
5	188.70 sec
6	166.02 sec
7	262.76 sec

- Line Balancing이 잘 진행됨
- 기계 가동하지 않을 시 사전 점검을 통해 장비 고장을 방지할 수 있음

05 최적화 Line Balancing & Makespan 최소화 검증

Plant Simulation으로 검증





- U자형 배치를 통해 인접 작업 간의 도움을 주도록 함
- 공장에서 하루 근무 9시간, 생산량은 500개라는 데이터를 얻음
- 개선된 표준시간과 작업 방식을 적용하여 시뮬레이션을 적용한 결과 126%의 생산량 증가를 유도함

.Models.Model

.Models.Model

Simulation time:9:00:00.0000

Object	Name	Mean Life Time	Throughput	TPH	Production	Transport	Storage	Value added	Portion
Drain	Part	2:45.226	1131	126	100.00%	0.00%	0.00%	63.55%	

Cumulated Statistics of the Parts which the Drain Deleted

Thank You

김재연(산업공학과), 김태훈(산업공학과),
신수연(산업공학과), 정호준(산업공학과)