



# FORMER 부품 제조 공장 상황을 고려한 생산계획 스케줄링 프로그램 제작

- 유전 알고리즘과 파이썬을 이용하여

경영과학 설계 - 7조

지도교수 : 모정훈 교수님  
2018147027 금종연  
2019147037 김주은  
2017232044 김진우  
2021147026 AYDIN ALI

## Background

### 배경 1 : 산업의 소금, 화스너

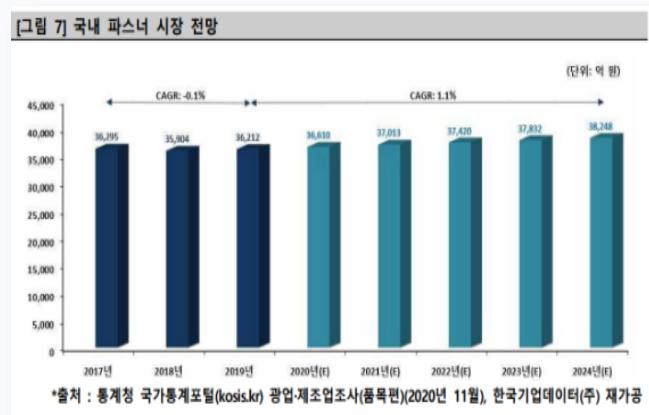
#### 화스너



부품을 결합하고 조립하는데 사용되는 부품

- ▶ 자동차, 선박, 전자, 기계, 중장비, 플랜트에 이르기까지 구조물의 지지력에 핵심 기능을 맡아 산업 점반에 필수로 쓰이는 가장 중요한 부품
- ▶ 우리 삶의 안전과도 직결된 부품

#### 시장 현황

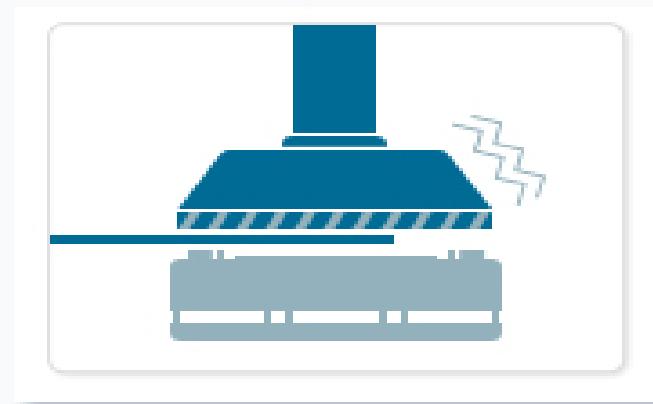


산업 구조에 따른 시장 성장세 지속 전망

- ▶ 자동차, 중장비 등 수요산업의 미약적인 발전에 따라 파스너의 대량 수요 발생 및 시장 성장 전망
- ▶ 표준 제품 대량생산 체제에서 고객이 요구하는 스펙을 충족하는 맞춤형 제품 생산 체제로 변화하고 있는 추세

### 배경 2 : FORMER 냉간 단조기의 등장

#### 화스너 제작 공정



제작 공정 : 재료 투입 → 단조 → 단조 이후

- ▶ 복잡한 형상의 제품을 정밀하게 대량생산화 가능 여부가 단조기술의 핵심 경쟁력
- ▶ 냉간단조 – 상온에서 가공, 수율 ↑, 정밀도 ↑

#### FORMER



산업용 볼트, 너트, 볼트 / 너트 파츠등을 생산하는 기계

- ▶ 단조 사업의 경우, 국내 기술 개발이 이루어지기 전에는 비싼 외국 설비에 의존하는 경향이 커짐
- ▶ 표준 제품 대량생산 체제에서 다품종 대량 생산을 경쟁력 있게 하는 것이 중요해지게 되면서 냉간 단조(FORMER) 기계의 필요성 증대

산업의 필수 소재인 화스너는 산업 수요 증가에 따라 수요량이 증가하고 있으며, 이에 맞는 화스너 제품 산업 수요가 증가하고 있다.

따라서 화스너에 대한 다품종 대량 생산을 경쟁력 있게 할 수 있게 하는 냉간 단조 기계의 필요성이 크게 증대되었다.

## Introduction

---

### 냉간 단조 기계 제작 회사 (효동 기계 공업)



- 주문 생산 방식에 의해 냉간 단조 기계를 생산하는 포머(Former)제작업체 (1988년 설립)
- 37년 동안 축적한 기술력을 기반으로 꾸준한 연구개발(R&D)을 추진, 단조 성형기계 독자 모델을 개발
- 연 매출 800억의 규모의 사업장으로 국내 포머 점유율 90% 이상을 차지하고 있는 냉간 단조 기계 제작의 선도 주자
- “고객 만족”이라는 기업 방침 아래 30년 넘게 고객의 다양성에 초점을 맞추어 옴

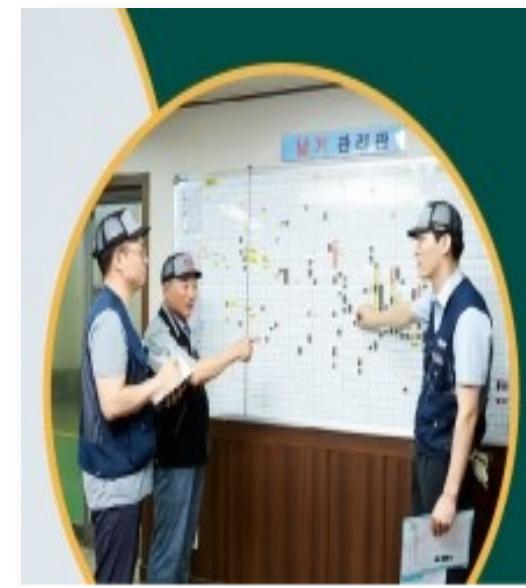
### 현재 생산 방식

HBF Series Specifications Bolt Former HBF-Series						
MODEL	Bolt Size (mm)	Cut-Off (mm)	Out put / min(mm)	KO (mm)	PKD (mm)	DIE Dia (mm)
HBF-506	M6 (1.0AF)	7x70	280	55	15	50
HBF-508	M8 (1.4AF)	9x95	250	70	15	60
HBF-510	M10 (1.7AF)	12x140	230	105	15	64
HBF-512	M12 (2.1AF)	12x140	150	155	15	65
HBF-514	M14 (2.4AF)	14x140	120	120	25	95
HBF-512L	M12 (2.1AF)	14x180	150	155	25	95
HBF-512LL	M12 (2.1AF)	14x250	120	210	30	85
HBF-512ULM	M12 (2.1AF)	14x350	70	310	50	100
		18x185	150	32	32	100
HBF-516	M16 (2.6AF)	18x300	100	20	37	110
HBF-518L	M18 (3.0AF)	18x390	55	360	50	110
HBF-520	M20 (3.2AF)	24x180	140	130	30	125
HBF-520L	M20 (3.2AF)	24x240	120	210	30	125
HBF-520UL	M20 (3.2AF)	24x460	55	360	55	130
HBF-522	M22 (3.6AF)	25x240	90	210	43	130
HBF-524	M24 (4.1AF)	30x270	90	190	52	150
HBF-524UL	M24 (4.1AF)	30x480	60	400	60	140
HBF-530	M30 (5.0AF)	33x375	80	300	50	160
HBF-530UL	M30 (5.0AF)	33x435	70	360	60	160
HBF-536	M36 (5.5AF)	36x360	70	300	50	170
HBF-536UL	M36 (5.5AF)	36x460	60	360	60	170
HBF-538	M38 (6.0AF)	42x395	55	300	50	190
HBF-538UL	M38 (6.0AF)	42x470	45	375	60	190
HBF-406	M6 (1.0AF)	7x70	280	55	15	50
HBF-408	M8 (1.4AF)	9x95	250	70	15	60
HBF-410	M10 (1.7AF)	12x140	230	105	15	64
HBF-410L	M10 (1.7AF)	12x170	150	155	15	65
HBF-412	M12 (2.1AF)	14x140	180	110	25	95
HBF-412L	M12 (2.1AF)	14x180	150	155	25	95
HBF-412ULM	M12 (2.1AF)	14x250	120	210	30	85
HBF-414	M14 (2.4AF)	14x250	100	310	50	100
HBF-414LL	M14 (2.4AF)	18x185	150	150	32	100
HBF-416	M16 (3.0AF)	18x270	100	240	32	110
HBF-416LL	M16 (3.0AF)	18x390	55	360	50	110
HBF-420	M20 (3.2AF)	24x180	140	130	30	125
HBF-420L	M20 (3.2AF)	24x460	55	360	55	130
HBF-422	M22 (3.6AF)	25x250	90	210	43	130

### 다품종 소량 생산

- 표준 제품 대량생산 체제가 아닌 고객이 요구하는 스펙을 충족하는 맞춤형 제품 생산 체제
- 납기 주문과 다양한 사양을 요구하는 고객의 주문
- 주문 생산 방식의 시스템에서는 일정 관리 문제가 고객만족의 첫 단추
- Job Shop 방식의 복잡한 생산 흐름

### 생산 계획 방식



#### 생산 계획

실무자의 경험에만 의존한 생산 계획 진행

#### 공정 관리

작업자들이 알아서 하던 부분을 공동 납기 현황 판을 이용해서 관리하여 생산 효율 개선



납기에 대한 가시성만 일정 부분 확보했을 뿐,  
구체적인 생산 계획 시스템은 마련되어있지 않은 현황

## Introduction

### 문제 현황 분석

- 생산 라인에서의 일정 계획에 대한 문제를 과학적인 방법 없이 실무자의 경험으로만 생산계획 및 관리를 함으로 인해 고객이 원하는 need의 즉각 반영이 어렵고 수요예측이 불확실
- 인원대비 매출액 (노동성장성), 노동 장비율 등이 현저히 떨어져 있는 실정이며 따라 현금흐름은 물론 공정 개선에 많은 어려움 보유
- 빈번한 납기지연과 기계의 이용률 그리고 인력 자원 등을 완벽히 활용하지 못하고 있는 현황
- 기계의 고장이나 긴급률, 불량, 작업 시간에 대한 대처가 신속하고 용이하게 이루어 질 수 없는 현황
- 따라 회사 경영 상태의 악화를 야기시키고, 납기로 인한 고객사에 대한 신뢰성이 떨어지고 있는 실정

→ 생산 효율성 확보 및 고객 서비스 수준 향상을 위해, 제조 공장내 상황을 고려한 체계적인 생산계획 스케줄링의 필요성 대두

### 왜 Scheduling을 바꿔야하니?

#### 스케줄링의 중요성

Scheduling은 조직의 비효율성을 최소화하고 고객만족을 최대화하는 생산 관리 활동 중의 하나

- ▶ 생산 스케줄링 구축 : 기업의 경쟁력 강화 및 이익 창출을 위한 중요한 과제
- ▶ 기업의 경쟁력 우위 확보를 위한 일종의 전술적 수단
- ▶ 단품종 소량 생산 방식을 취하고 있는 주문 생산 방식의 시스템에서는 일정 관리 문제가 고객만족의 핵심 요인



## ■ Problem statement / Key idea



### 유전 알고리즘을 이용한 스케줄링 프로그램 제작

생산 효율성 확보 / 수익성 방어 및 고객 서비스 수준 향상을 위해, FORMER 부품 제조 공장 상황을 고려한 체계적인 생산계획 스케줄링 프로그램을 마련하는 것이 중요



### 핵심 고려 요인 및 문제 정의

#### FORMER 제작 과정

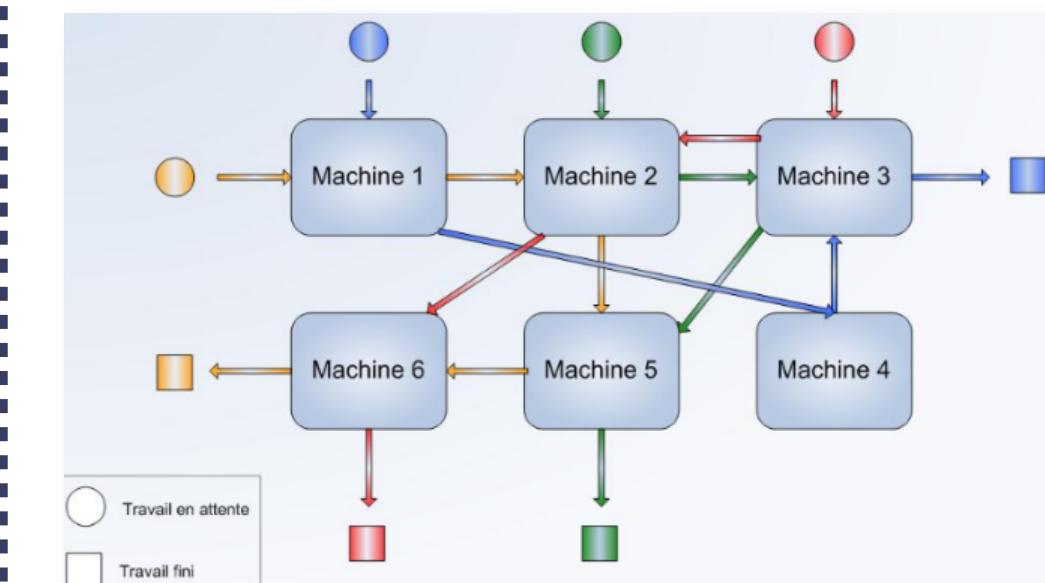
#### -납기 관리, 부품 공정 스케줄링-



- 생산 패러다임의 전환으로 생산량 증대를 목적으로 하는 스케줄링 보다 납기관리의 스케줄링의 중요성이 증대된 현황
- 다양한 종류의 부품으로 이루어져, 좌측과 같은 조립 흐름으로 FORMER라는 냉간 단조기를 생산
- 장비의 생산수준을 결정짓는 부분은 부품 가공에 대한 scheduling 문제라고 해도 과언이 아닐 정도로 부품에 대한 일정관리 문제가 매우 중요한 현황

#### 부품 제조 방식

#### -Job shop-



- Job Shop 생산 방식 : 부품 가공 파트의 세부 공정은 각기 다른 원재료가 투입되어 각각의 가공 시간과 공정에 따라서 부품이 완성되는 Job Shop 형태로 이루어져 있다.

## Problem statement / Key idea

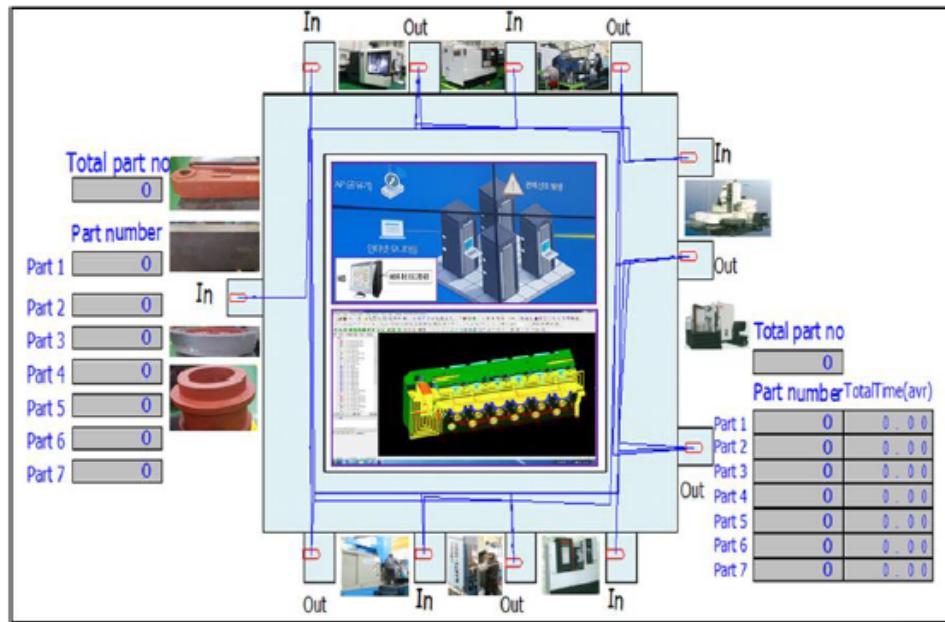
### 핵심 고려 요인 및 문제 정의

공장 내 배치 상황

- 공정간 이동시간, 기계 대수 -

공정 종류

- 부품별 공정도, 최대 공정 개수, 공정별 프로세스 순서 & 타임 -



- 공장내 부품 설비는 위의 그림과 같은 형태로 배치되어 있다.
- 모든 기계들은 어떠한 가공 시점에서 두개 이상의 부품을 동시에 가공할 수 없고 가공 도중에 다른 부품의 가공을 시작 할 수 없다.
- 하나의 공정이 끝나고 다른 공정으로 이동시, 이동시간이 발생한다.



Part	Part Name	Process			Image
		L	M	R	
Part1	FLANGE				
Part2	SHAFT	L	M	R	
Part3	GUIDE	CNC	M	CNC	
Part4	BRACKET	M	G	MCT	
Part5	BOSS	M	L	R	
Part6	KO CONNECT	L	B	R	
Part7	SCREW	CNC	M		
Part8	PIN	L	M	R	
Part9	PIPE SCREW	L	M	R	
Part10	LOCK NUT	M	R	L	

MODEL		18x120
HBP-618		

- 공정 종류 : 선반, 밀링, 보링, CNC, MCT, G, 레디얼
- 각 부품마다 거쳐야 하는 공정의 개수는 2 ~ 6개
- 각 부품에 대한 공정 순서 및 공정 시간은 상이, 해당 프로세스 순서는 공정도에 기록되어있음.

표 2-2] HBP-618 공정별 가공시간 (표준시간)

Part	Part Name	Process Time (단위:H)			Total
		Part 1	FLANGE	1.8	
					3.3

### 주문 및 제작 방식

### MTO 방식, 주문 제품별로 부품 제작 및 생산하는 체계

#### HBP Series 볼트파츠 포머 Hyodong Bolt Parts Former

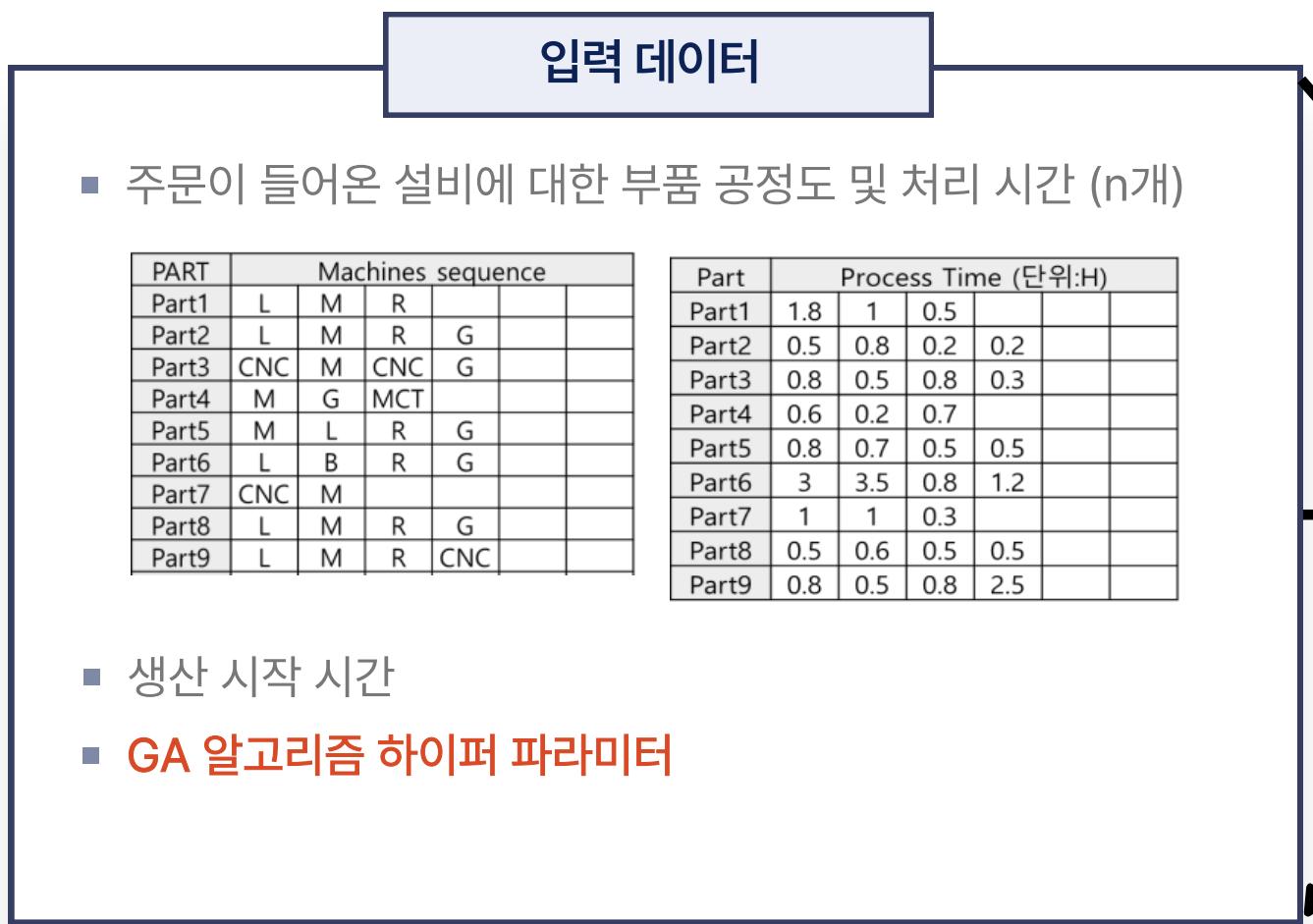
효동의 볼트파츠 포머는 5단에서 8단까지 제작이 가능하며 고객의 요구에 따라 다양한 타입의 기종도 개발이 가능합니다. 특히 트랜스퍼의 경우 카세트 형식으로 되어있어 오픈, 턴, 유니버설 척 등을 조합하여 사용할 수 있어 다양한 제품군을 성형하는데 최적화된 기종입니다.

단수 Station	최대 절단경 x 길이 Max. Cut-off Dia x Length	최대 KO Max. KO mm	최대 압조력 Forging Lord kN	생산수 Speed PPM
5 ~ 8	Ø 50 X 450 mm	480 mm	13,000	40 ~ 250
HBP-511	11x70	18	70	26
HBP-515	15x105	22	85	26
HBP-518	18x120	25	110	35
HBP-523	23x160	32	130	40
HBP-523L	23x175	32	175	45
HBP-527	27x190	40	150	45

- 고객의 요구에 따라 다양한 형태와 크기로 주문제작되는 방식
- 주문 당시 인도 납기를 고객에게 전달 받거나, 예상 납기일을 전달하는 체계
- 관리되는 납기의 종류는 많지만, 제조 부품 공정의 경우 부품의 제작이 모두 완료되어야 FORMER 조립에 들어갈 수 있기에 마감 일자가 거의 같은 특징

## Problem statement / Key idea

### 스케줄링 프로그램 안내



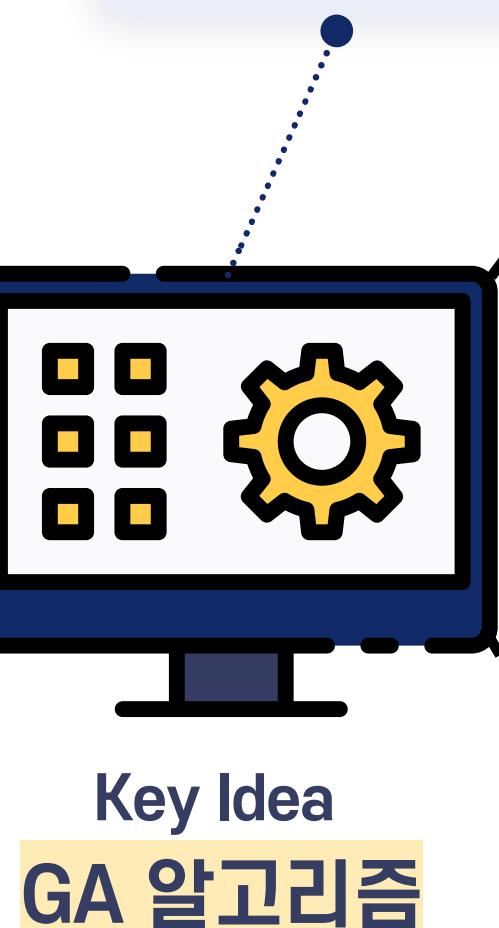
### 공장 내 상황 변동 조정 가능

- 공정간 이동시간
- 기계 대수
- 가동률 \*

### 제조업 가동률 \*

사업체가 갖고 있는 설비가 노동, 생산효율 등 일정한 조건 하에서 정상적으로 가동했을 때 생산 할 수 있는 최대 생산능력에 대한 실제 생산량의 비율 (%)

중소 제조업 평균 가동률 : 71.5 % (중소기업중앙회)



### 출력 데이터

- 입력 데이터에 대한 각 공정 설비의 Gantt chart



- 생산 종료 시간 (2022-01-06 08:14:00)
- 각 공장 설비별 최적 작업 순서, 부품별 완료 시점 및 순서
- Optimal makespan (113.5(h))

## Model formulation

### 목적 함수

- 총 생산시간(Makespan) 최소화

### 결정 변수

- 각 기계별 작업 처리 순서

### 가정

- 각 작업은 사전에 정의된 공정 순서에 따라 처리된다.
- 모든 기계들은 어떠한 가공 시점에서 두개 이상의 부품을 동시에 가공 할 수 없고 가공 도중에 다른 부품의 가공을 시작 할 수 없다
- 각 작업의 공정 간 이동거리는 6분이다.
- 모든 기계는 고장 없이 연속적으로 이용 가능하다.
- 각 기계가 처리할 수 있는 공정은 정해져있다.
- 각각의 부품 가공시간은 공정 및 부품별로 다르다.

## Improvement

### Module 2 개선 내용

## Module 2 고도화 : 공장 내 상황 반영 + 모듈의 특성 활용 증명을 위한 Parameter 변경 작업 진행

**제조 공장 내 세부 사항 반영**

- 기계 대수
- 기계 가동 시간
- 작업 수

**1. 기계 대수 반영**

**기존**

- 공정 당 1개의 머신

**개선**

- 각 공정 당 실제 기계 수 반영

- L\_num = 8    - CNC\_num = 1  
- M\_num = 3    - MCT\_num = 7  
- R\_num = 2    - B\_num = 7  
- G\_num = 5

**2. 기계 가동 시간 반영**

**기존**

- 반영 안 함 - 24시간 풀 가동

**개선**

- 실제 가동률

- 중소기업중앙회에서 발표한 중소 제조업 평균 가동률 (71.5%) 반영

**3. 작업 수 증가**

**기존**

- PJT 1개 (작업 30개) 이용

**개선**

- PJT 4개 (작업 120개) 이용

- Case 1 = 40 jobs (PJT 1)  
- Case 2 = 80 jobs (PJT 1&2)  
\* Process Time이 너무 긴 3개의 part 제거  
- Case 3 = 117 jobs \* (PJT 1&2&3)  
- Case 4 = 120 jobs (PJT 1&2&3)

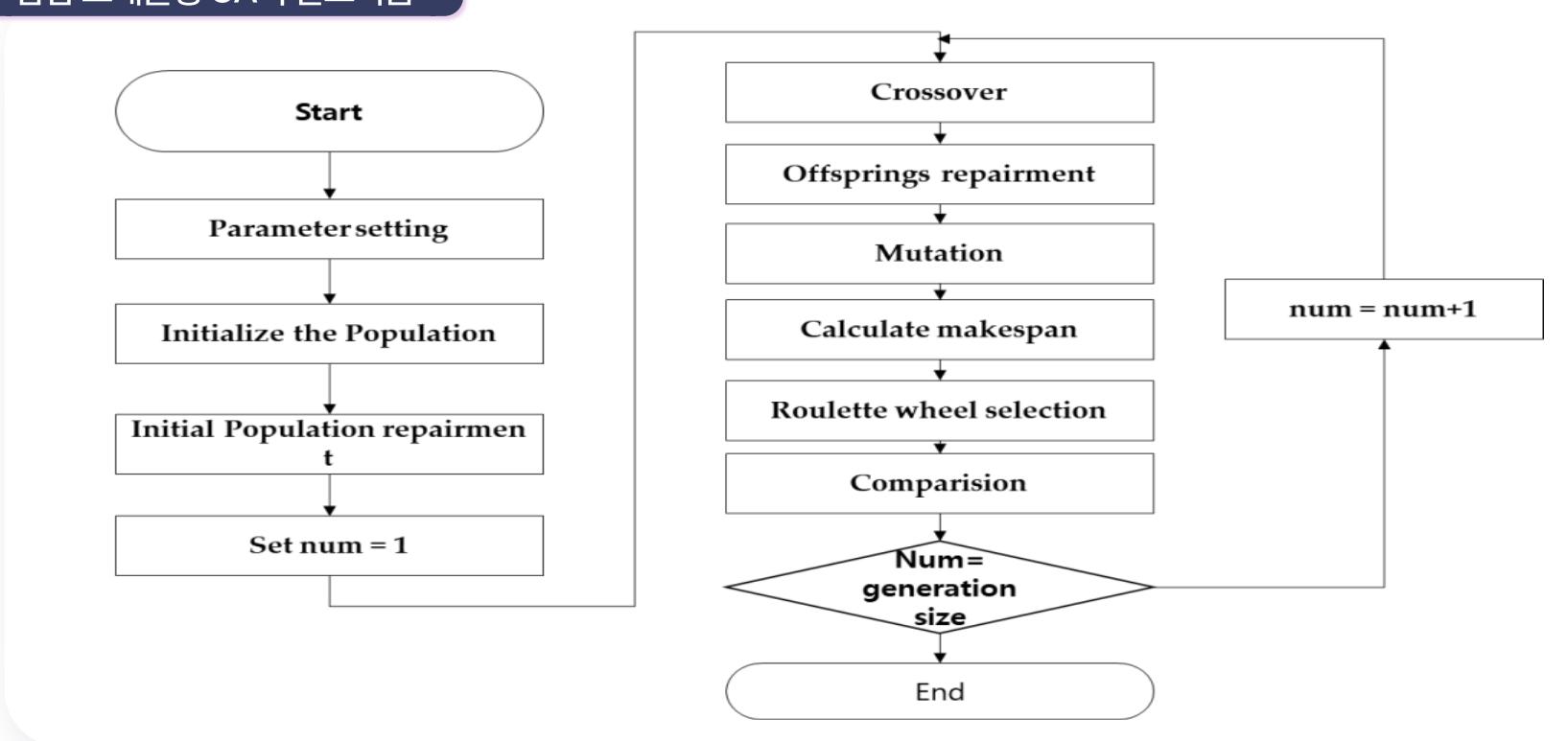
## Method / Experiment

### GA 알고리즘

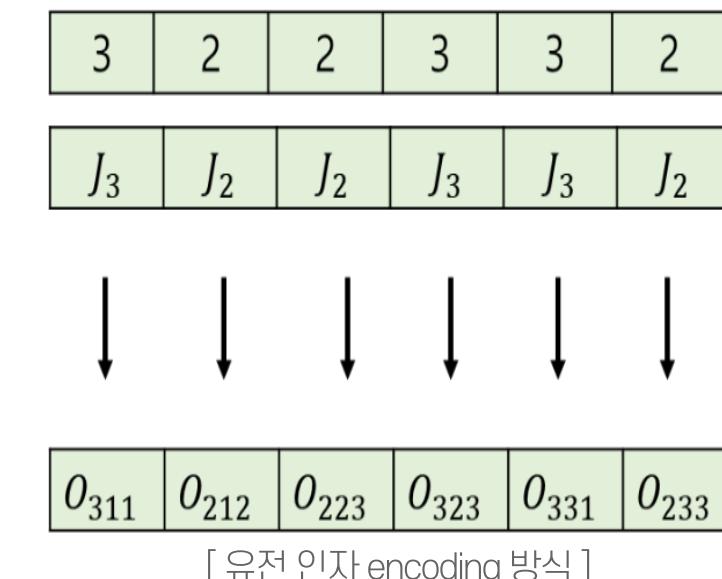
- Job Shop Scheduling 문제는 NP-hard 문제로, 일반적으로 휴리스틱(heuristics) 또는 메타휴리스틱(Metaheuristics) 알고리즘을 적용하여 근사해를 구한다
- GA는 생물이 환경에 적응하면서 진화해가는 모습을 모방하여 최적해를 찾아내는 검색 방법으로 이론적으로 전역 최적점을 찾을 수 있으며, 수학적으로 명확하게 정의되지 않은 문제에도 적용 가능한 알고리즘

- 따라서 스케줄링과 같이 단계적으로 정의된 문제 내에서 여러 번의 시행을 거쳐 최적의 순서를 찾기에 용이한 특성
- GA는 해당 문제에 대한 적합도의 식을 얼마나 정교하게 설정하는 정도에 따라 그 성능을 결정하는 특성 보유

#### 집합 스케줄링 GA의 알고리즘

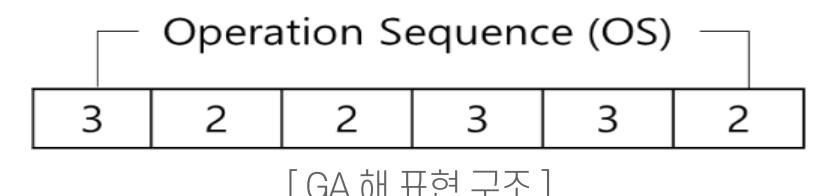


#### 해 표현 방법



■ 선행연구(Zhang et al.(2017) 등)에서 GA에 주로 사용하는 해 표현 방법 활용

■ 염색체는 공정의 처리순서를 표현하기 위해 공정의 수만큼 반복되는 형태를 갖는다.  
**(염색체의 크기 = 공정의 개수)**



# Method / Experiment

## GA 알고리즘

### Data 및 Hyperparameter 입력

#### ■ 데이터

- 1. Machines sequence    2. Processing time

#### ■ 입력값

- |                    |                            |                                 |
|--------------------|----------------------------|---------------------------------|
| 1. Moving_time (H) | 4. Crossover_rate          | 7. Generation                   |
| 2. Start_time      | 5. Mutation_rate           | <b>8. Capa_utilization_rate</b> |
| 3. Population_size | 6. Mutation_selection_rate | <b>9. mc_num</b>                |

#### ■ 소스 코드

```

pt_tmp=pd.read_excel("JSP_dataset.xlsx",sheet_name="Processing Time",index_col =[0])
ms_tmp=pd.read_excel("JSP_dataset.xlsx",sheet_name="Machines Sequence",index_col =[0])

start_time = datetime.datetime.strptime("2022-01-01 14:44", "%Y-%m-%d %H:%M")
moving_time = 0.1
Capa_utilization_rate = 0.715
max_process=len(pt_tmp.columns)
num_job=len(pt_tmp)
#num_mc=7
L_num = 8
M_num = 3
R_num = 2
G_num = 5
CNC_num = 5
MCT_num = 1
MCT_num = 7
B_num = 7
num_mc = L_num+M_num+R_num+G_num+CNC_num+MCT_num+B_num
num_gene=max_process*num_job - pt_tmp.isnull().sum()
num_gene=max_process*num_job - pt_tmp.sum()

pt=[list(map(float,pt_tmp.iloc[i][0:max_process-pt_tmp.iloc[i].isnull().sum()])) for i in range(num_job)]
ms=[list(map(float,ms_tmp.iloc[i][0:max_process-ms_tmp.iloc[i].isnull().sum()])) for i in range(num_job)]

population_size = 50
crossover_rate = 0.8
mutation_rate = 0.8
mutation_selection_rate = 0.5
num_mutation_jobs = round(num_gene*mutation_selection_rate)
generation = 1000

Tbest=99999999999999
current_time = datetime.datetime.now()

```

### 초기값 생성

#### ■ 적용 방법

입력한 Population의 크기에 따라 전체 중 80%는 Random 방식으로 초기 값을 생성하고, 10%는 LPT, 나머지 10%는 SPT 방식으로 초기값을 생성하여 함께 사용하였다. 각각의 염색체는 공정 수의 유전자를 가지고 있다.

#### ■ 소스 코드

```

best_list,best_obj=[],[]
population_list=[]
makespan_record=[]
for i in range(int(population_size*0.8)):
    nxm_random_num=list(np.random.permutation(num_gene))
    population_list.append(nxm_random_num)
    for j in range(num_gene):
        population_list[i][j]=population_list[i][j]%(num_job)
for i in range(int(population_size*0.1)):
    population_list.append(LPT_IP)
    population_list.append(SPT_IP)

```

### 초기값 조정

#### ■ 적용 방법

Random 방식으로 생성된 초기값들에서 각 작업의 공정이 사전에 정의된 개수 만큼 처리될 수 있도록 수정한다.

#### ■ 소스 코드

```

for s in range(population_size):
    job_count={}
    larger,less=[], []
    for i in range(num_job):
        process_num_i = len(ms_tmp.iloc[i])-ms_tmp.iloc[i].isnull().sum()
        if i in population_list[s]:
            count=population_list[s].count(i)
            pos=population_list[s].index(i)
            job_count[i]=[count,pos]
        else:
            count=0
            job_count[i]=[count,0]
        if count>process_num_i:
            larger.append(i)
        elif count<process_num_i:
            less.append(i)
    for k in range(len(larger)):
        chg_job=larger[k]
        process_num_chg_job = len(ms_tmp.iloc[chg_job])-ms_tmp.iloc[chg_job].isnull().sum()
        while job_count[chg_job][0]>process_num_chg_job:
            for d in range(len(less)):
                process_num_d = len(ms_tmp.iloc[less[d]])-ms_tmp.iloc[less[d]].isnull().sum()
                if job_count[less[d]][0]<process_num_d:
                    population_list[s][job_count[chg_job][1]]=less[d]
                    job_count[chg_job][1]=population_list[s].index(chg_job)
                    job_count[less[d]][0]=job_count[chg_job][0]-1
                    job_count[less[d]][0]=job_count[less[d]][0]+1
                if job_count[chg_job][0]==process_num_chg_job:
                    break

```

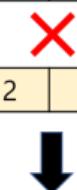
# Method / Experiment

## GA 알고리즘

### Crossover

#### ■ 적용 방법

Parent 1 [3 2 1 2 3 2 1 2]



Parent 2 [2 1 3 2 1 2 2 3]

Random 방식으로 생성된 초기값들에서 각 작업의 공정이 사전에 정의된 개수 만큼 처리될 수 있도록 수정한다.

Offspring 1 [3 2 3 2 1 2 1 2]

Offspring 2 [2 1 1 2 3 2 2 3]

#### ■ 소스 코드

```
parent_list=copy.deepcopy(population_list)
offspring_list=copy.deepcopy(population_list)
S=list(np.random.permutation(population_size))

for m in range(int(population_size/2)):
    crossover_prob=np.random.rand()
    if crossover_rate>=crossover_prob:
        parent_1=population_list[S[2*m]][:]
        parent_2=population_list[S[2*m+1]][:]
        child_1=parent_1[:]
        child_2=parent_2[:]
        cutpoint=list(np.random.choice(num_gene, 2, replace=False))
        cutpoint.sort()

        child_1[cutpoint[0]:cutpoint[1]]=parent_2[cutpoint[0]:cutpoint[1]]
        child_2[cutpoint[0]:cutpoint[1]]=parent_1[cutpoint[0]:cutpoint[1]]
        offspring_list[S[2*m]]=child_1[:]
```

### Offspring 조정

#### ■ 적용 방법

Crossover로 생성된 자손들에서 각 작업의 공정이 사전에 정의된 개수 만큼 처리될 수 있도록 수정한다

#### ■ 소스 코드

```
for m in range(population_size):
    job_count={}
    larger,less=[], []
    for i in range(num_job):
        process_num_i = len(ms_tmp.iloc[i]) - ms_tmp.iloc[i].isnull().sum()
        if i in offspring_list:
            count=offspring_list[m].count(i)
            pos=offspring_list[m].index(i)
            job_count[i]=[count,pos]
        else:
            count=0
            job_count[i]=[count,0]

        if count>process_num_i:
            larger.append(i)
        elif count<process_num_i:
            less.append(i)

    for k in range(len(larger)):
        chg_job=larger[k]
        process_num_chg_job = len(ms_tmp.iloc[chg_job]) - ms_tmp.iloc[chg_job].isnull().sum()
        while job_count[chg_job][0]>process_num_chg_job:
            for d in range(len(less)):
                process_num_d = len(ms_tmp.iloc[less[d]]) - ms_tmp.iloc[less[d]].isnull().sum()
                if job_count[less[d]][0]<process_num_d:
                    offspring_list[m][job_count[chg_job][1]]=less[d]
                    job_count[chg_job][1]=offspring_list[m].index(chg_job)
                    job_count[chg_job][0]=job_count[chg_job][0]-1
                    job_count[less[d]][0]=job_count[less[d]][0]+1
                    if job_count[chg_job][0]==process_num_chg_job:
                        break
```

### Mutation

#### ■ 적용 방법

전역 최적화를 위해 Mutation rate에 따라 둘 연변이를 생성하여 활용한다

Mutation 2 → 5 → 6 → 2

Index 1 2 3 4 5 6

Cromosome [4 2 1 5 3 6]

New [4 6 1 5 2 3]

#### ■ 소스 코드

```
for m in range(len(offspring_list)):
    mutation_prob=np.random.rand()
    if mutation_rate >= mutation_prob:
        m_chg=list(np.random.choice(num_gene, num_mutation_jobs, replace=False))
        t_value_last=offspring_list[m][m_chg[0]]
        for i in range(num_mutation_jobs-1):
            offspring_list[m][m_chg[i]]=offspring_list[m][m_chg[i+1]]
            offspring_list[m][m_chg[num_mutation_jobs-1]]=t_value_last
```

# Method / Experiment

## GA 알고리즘

### 선택

#### ■ 적용 방법

각 염색체가 형성한 스케줄의 Makespan을 계산하여 저장한 뒤, Roulette wheel selection 방법을 이용하여 세대를 구성한다.

다음으로 세대 내에서 가장 Makespan이 낮은 염색체를 선택하여 저장한다.

#### ■ 소스 코드

```

total_chromosome=copy.deepcopy(parent_list)+copy.deepcopy(offspring_list)
chrom_fitness,chrom_fit,[],[]
total_fitness=0
for m in range(population_size*2):
    j_keys=[j for j in range(num_job)]
    key_count={key:0 for key in j_keys}
    j_count={key:0 for key in j_keys}
    L_keys=["1"+ "_" +str(j+1) for j in range(L_num)]
    M_keys=["2"+ "_" +str(j+1) for j in range(M_num)]
    R_keys=["3"+ "_" +str(j+1) for j in range(R_num)]
    G_keys=["4"+ "_" +str(j+1) for j in range(G_num)]
    CNC_keys=["5"+ "_" +str(j+1) for j in range(CNC_num)]
    MCT_keys=["6"+ "_" +str(j+1) for j in range(MCT_num)]
    B_keys=["7"+ "_" +str(j+1) for j in range(B_num)]
    m_keys = L_keys+M_keys+R_keys+G_keys+CNC_keys+MCT_keys+B_keys
    m_count={key:0 for key in m_keys}

    for i in total_chromosome[m]:
        gen_t=float(pt[i][key_count[i]])/Capa_utilization_rate
        gen_m=int(ms[i][key_count[i]])
        j_count[i]=j_count[i]+gen_t+moving_time
        L_min = 999999
        M_min = 999999
        R_min = 999999
        G_min = 999999
        CNC_min = 999999
        MCT_min = 999999
        B_min = 999999

        if gen_m == 1:
            for k in range(1,L_num+1):
                L_temp = m_count["1_"+str(k)]
                if L_temp < L_min:
                    L_min = L_temp
                    index = "1_"+str(k)
                else:
                    continue
                m_count[index]=m_count[index]+gen_t

        elif gen_m == 4:
            for k in range(1,G_num+1):
                G_temp = m_count["4_"+str(k)]
                if G_temp < G_min:
                    G_min = G_temp
                    index = "4_"+str(k)
                else:
                    continue
                m_count[index]=m_count[index]+gen_t

        elif gen_m == 2:
            for k in range(1,M_num+1):
                M_temp = m_count["2_"+str(k)]
                if M_temp < M_min:
                    M_min = M_temp
                    index = "2_"+str(k)
                else:
                    continue
                m_count[index]=m_count[index]+gen_t

        elif gen_m == 5:
            for k in range(1,CNC_num+1):
                CNC_temp = m_count["5_"+str(k)]
                if CNC_temp < CNC_min:
                    CNC_min = CNC_temp
                    index = "5_"+str(k)
                else:
                    continue
                m_count[index]=m_count[index]+gen_t

        elif gen_m == 3:
            for k in range(1,R_num+1):
                R_temp = m_count["3_"+str(k)]
                if R_temp < R_min:
                    R_min = R_temp
                    index = "3_"+str(k)
                else:
                    continue
                m_count[index]=m_count[index]+gen_t

        elif gen_m == 6:
            for k in range(1,MCT_num+1):
                MCT_temp = m_count["6_"+str(k)]
                if MCT_temp < MCT_min:
                    MCT_min = MCT_temp
                    index = "6_"+str(k)
                else:
                    continue
                m_count[index]=m_count[index]+gen_t

    total_fitness+=chrom_fitness[m]
    chrom_fitness.append(1/makespan)
    chrom_fit.append(makespan)
    total_fitness+=chrom_fitness[m]

pk,qk,[],[]
for i in range(population_size*2):
    pk.append(chrom_fitness[i]/total_fitness)
for i in range(population_size*2):
    cumulative=0
    for j in range(0,i+1):
        cumulative=cumulative+pk[j]
    qk.append(cumulative)

selection_rand=np.random.rand() for i in range(population_size):
    if selection_rand[i]<qk[0]:
        population_list[i]=copy.deepcopy(total_chromosome[0])
    else:
        for j in range(0,population_size*2-1):
            if selection_rand[i]>qk[j] and selection_rand[i]<qk[j+1]:
                population_list[i]=copy.deepcopy(total_chromosome[j+1])
                break

for i in range(population_size*2):
    if chrom_fit[i]<Tbest_now:
        Tbest_now=chrom_fit[i]
        sequence_now=copy.deepcopy(total_chromosome[i])

if Tbest_now<Tbest:
    Tbest=Tbest_now
    sequence_best=copy.deepcopy(sequence_now)

makespan_record.append(Tbest)

```

## Method / Experiment

### 결과 출력

- Optimal Sequence, Optimal makespan, End time, Gantt chart을 출력하여 최적 스케줄링과 최적 Makespan을 활용한다.
- 나아가 Generation 별 최소 Makespan을 비교하여 최적값을 찾아가는 과정을 확인한다.

### 결과값 시각화

Case 1

40 jobs (Project 1)

Start time

2022-01-01 14:44

#### ■ 공정 순서 데이터

1	PART	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6
2	Part1	1	2	3			
3	Part2	1	2	3	4		
		⋮					
40	Part39	7	3	7			
41	Part40	7	3				

#### ■ 가공 시간 데이터

1	PART	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6
2	Part1	1.8	1	0.5			
3	Part2	0.5	0.8	0.2	0.2		
		⋮					
40	Part39	10.5	1.2	13.8			
41	Part40	14	2.5				

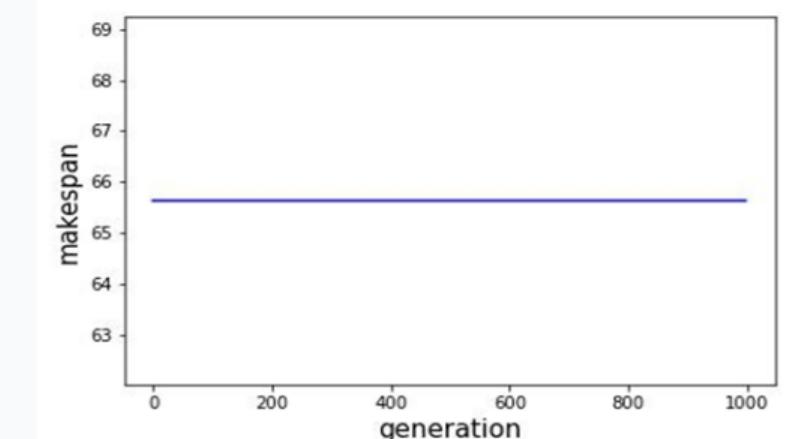
#### ■ 간트 차트



#### ■ Optimal makespan, End time

- Optimal makespan : 65.63 (H)
- End time : 2022-01-04 08:22:05

#### ■ Generation 별 Makespan



## Method / Experiment

결과 출력

결과값 시각화

Case 2

80 jobs (Project 1 & 2)

Start time

2022-01-01 14:44

▪ 공정 순서 데이터

1	PART	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6
2	Part1	1	2	3			
3	Part2	1	2	3	4		
		⋮					
80	Part39	7	3	7			
81	Part40	7	3				

▪ 가공 시간 데이터

1	PART	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6
2	Part1	1.8	1	0.5			
3	Part2	0.5	0.8	0.2	0.2		
		⋮					
80	Part39	12	1.2	20			
81	Part40	16	4				

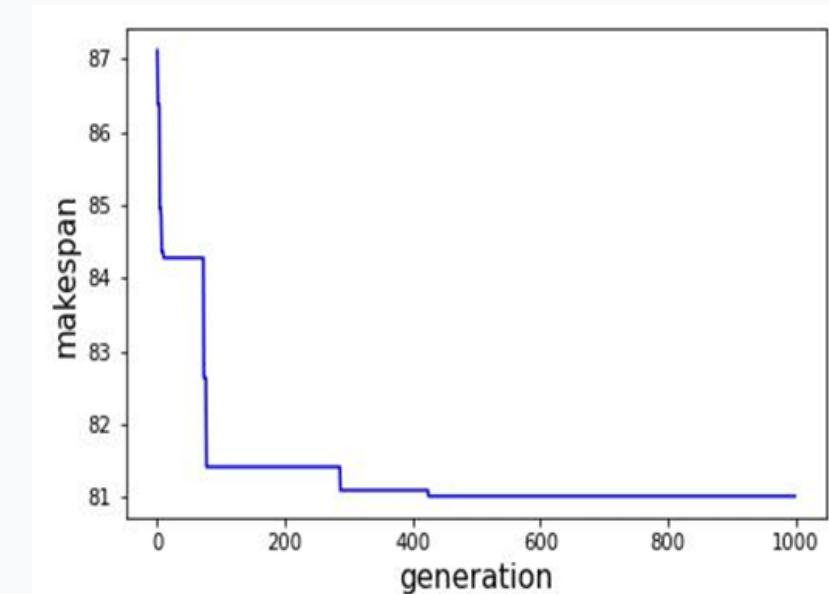
▪ 간트 차트



▪ Optimal makespan, End time

- Optimal makespan : 81.02 (H)
- End time : 2022-01-04 23:45:10

▪ Generation 별 Makespan



## Method / Experiment

### 결과 출력

#### 결과값 시각화

Case 3

117 jobs

(Project 1 &amp; 2 &amp; 3에서 Process Time이 너무 긴 3개의 part 제거)

Start time

2022-01-01 14:44

#### ▪ 공정 순서 데이터

1	PART	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6
2	Part1	1	2	3			
3	Part2	1	2	3	4		

⋮

118	Part37	1	2	3	1	5
119	Part38	2	4	6	3	
120	Part39	7	5	3	7	X
121	Part40	7	3			

#### ▪ 가공 시간 데이터

1	PART	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6
2	Part1	1.8	1	0.5			
3	Part2	0.5	0.8	0.2	0.2		

⋮

118	Part37	20	10	6	8	5
119	Part38	100	50	205	10	
120	Part39	150	20	240		X
121	Part40	205	20			

#### ▪ 간트 차트

Job shop Schedule

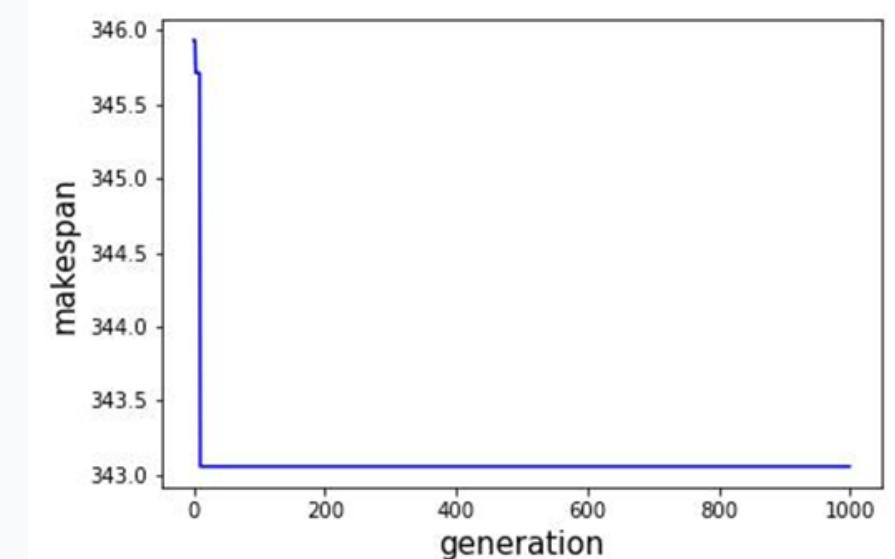
1w 1m 6m YTD 1y all



#### ▪ Optimal makespan, End time

- Optimal makespan : 343.06 (H)
- End time : 2022-01-15 21:47:26

#### ▪ Generation 별 Makespan



## Method / Experiment

결과 출력

결과값 시각화

Case 4

120 jobs (Project 1 & 2 & 3)

Start time

2022-01-01 14:44



기존 수기로 진행하던 방식에 비해 신속성, 신뢰성, 가시성을 확보하는 효과 창출

▪ 공정 순서 데이터

1	PART	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6
2	Part1	1	2	3			
3	Part2	1	2	3	4		

⋮

118	Part37	1	2	3	1	5
119	Part38	2	4	6	3	
120	Part39	7	3	7		
121	Part40	7	3			

▪ 가공 시간 데이터

1	PART	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6
2	Part1	1.8	1	0.5			
3	Part2	0.5	0.8	0.2	0.2		

⋮

118	Part37	20	10	6	8	5
119	Part38	160	50	205	10	
120	Part39	150	20	240		
121	Part40	205	30			

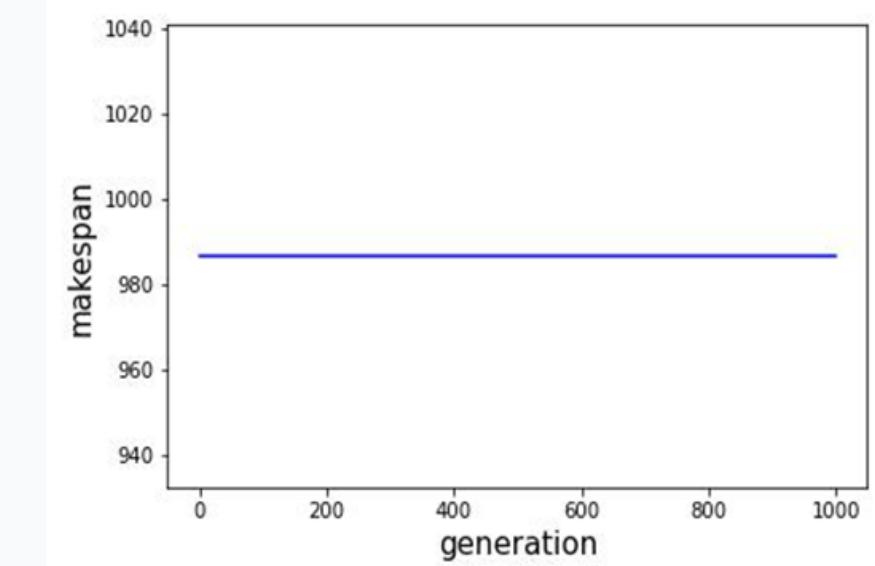
▪ 간트 차트



▪ Optimal makespan, End time

- Optimal makespan : 986.61 (H)
- End time : 2022-02-11 17:20:50

▪ Generation 별 Makespan



## Method / Experiment

### 방법론 비교 (우선순위규칙)

#### 최소가공시간 우선 규칙 (SPT rule)

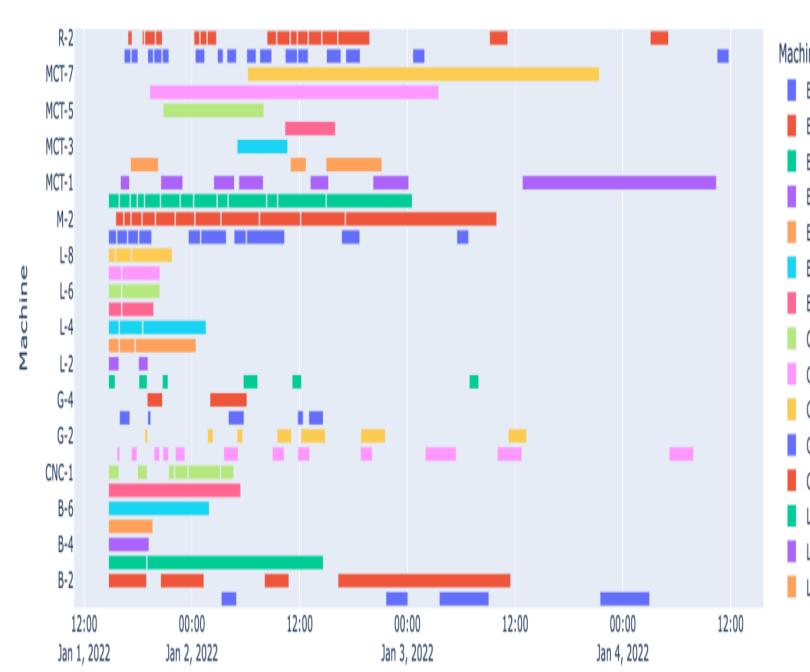
- 부품 투입이 순서와 관계없이 가공시간이 가장 짧은 부품을 먼저 처리하는 형태. (목적 관리 지표 : job의 체류시간 최소화)

Case 1

Start time

2022-01-01 14:44

#### 간트 차트



#### Optimal makespan, End time

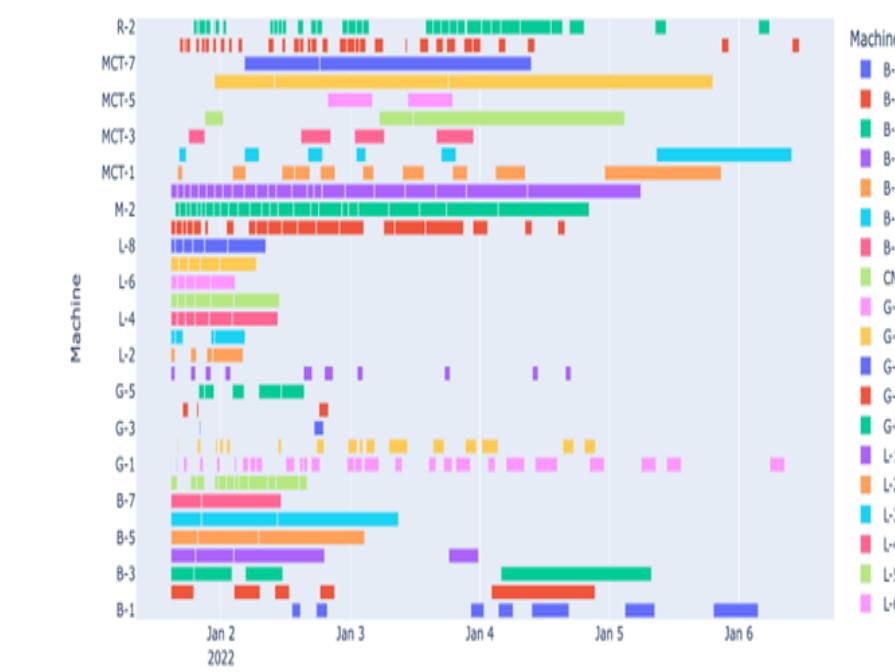
Optimal makespan  
78.12 (H)  
End time  
2022-01-04 20:51:23

Case 2

Start time

2022-01-01 14:44

#### 간트 차트



#### Optimal makespan, End time

Optimal makespan  
129.23 (H)  
End time  
2022-01-06 23:57:55



Case 1, Case 2의 경우 모두, (GA 알고리즘을 사용하였을 때의 Makespan) < (SPT rule을 사용했을 때의 Makespan)

## Method / Experiment

방법론 비교 (우선순위규칙)

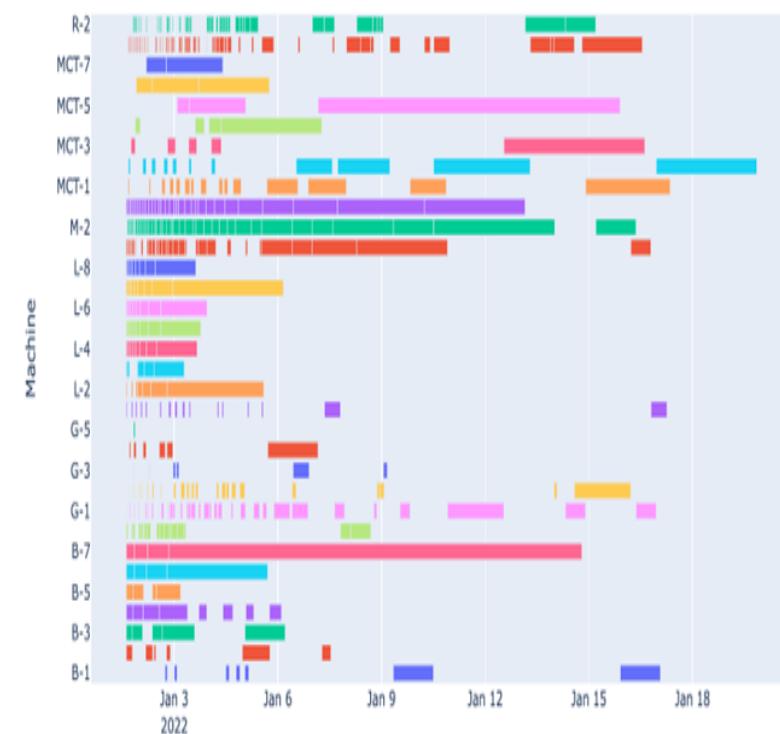
최소가공시간 우선 규칙 (SPT rule)

Case 3

Start time

2022-01-01 14:44

■ 간트 차트



■ Optimal makespan, End time

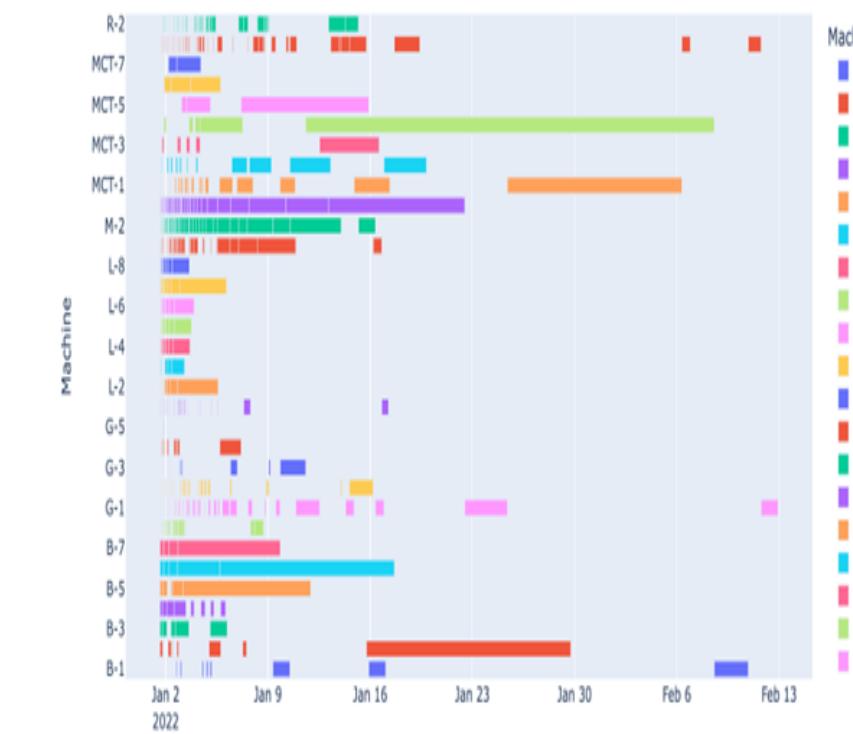
Optimal makespan  
454.21 (H)  
End time  
2022-01-20 12:56:25

Case 4

Start time

2022-01-01 14:44

■ 간트 차트



■ Optimal makespan, End time

Optimal makespan  
1098.42 (H)  
End time  
2022-02-16 09:09:23



Case 3, Case 4의 경우 모두, (GA 알고리즘을 사용하였을 때의 Makespan) < (SPT rule을 사용했을 때의 Makespan)

## Method / Experiment

### 방법론 비교 (우선순위규칙)

### 최대 가공 시간 우선 규칙 (LPT rule)

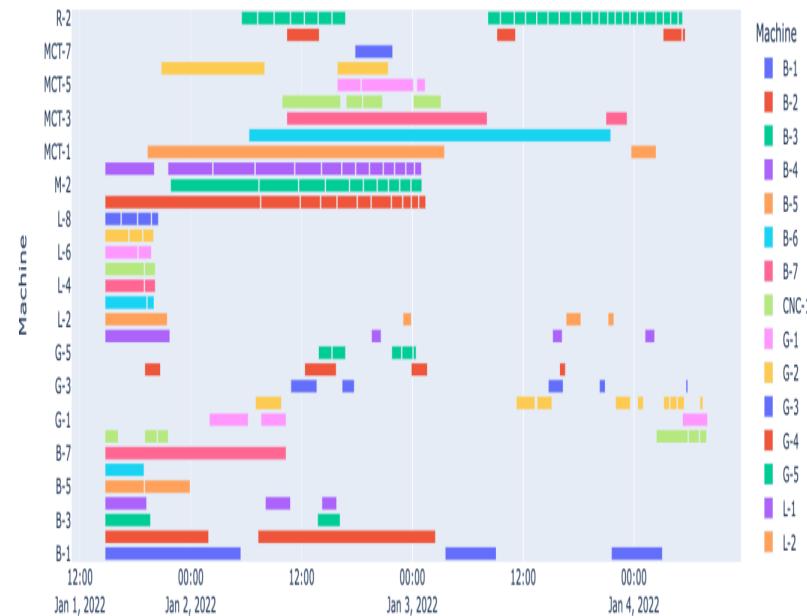
- 부품 투입이 순서와 관계없이 가공시간이 가장 긴 부품을 먼저 처리하는 형태. (목적 관리 지표 : job의 체류시간 최소화)

Case 1

Start time

2022-01-01 14:44

#### 간트 차트



#### Optimal makespan, End time

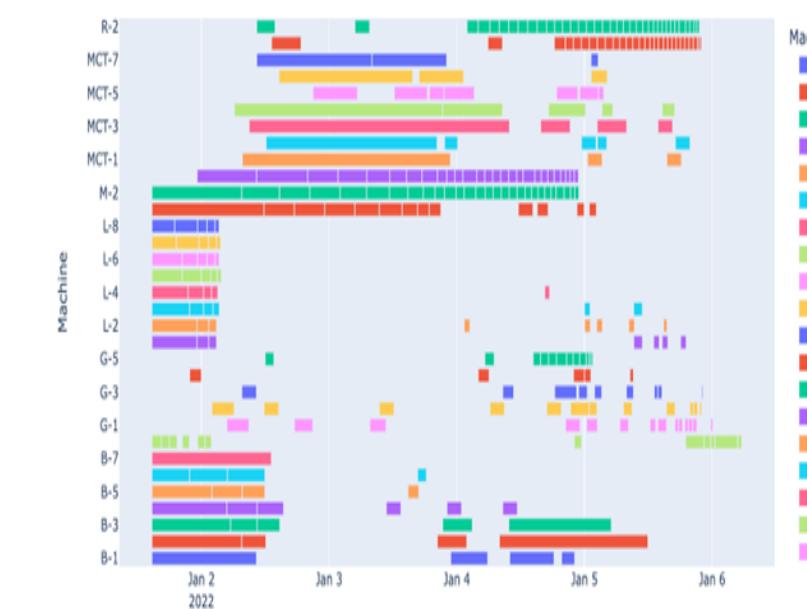
Optimal makespan  
73.32 (H)  
End time  
2022-01-04 16:03:38

Case 2

Start time

2022-01-01 14:44

#### 간트 차트



#### Optimal makespan, End time

Optimal makespan  
110.99 (H)  
End time  
2022-01-06 05:43:22



Case 1, Case 2의 경우 모두, (GA 알고리즘을 사용하였을 때의 Makespan) < (LPT rule을 사용했을 때의 Makespan)

## Method / Experiment

### 방법론 비교 (우선순위규칙)

#### 최대 가공 시간 우선 규칙 (LPT rule)

- 부품 투입이 순서와 관계없이 가공시간이 가장 긴 부품을 먼저 처리하는 형태. (목적 관리 지표 : job의 체류시간 최소화)

Case 3

Start time

2022-01-01 14:44

- 간트 차트



- Optimal makespan, End time

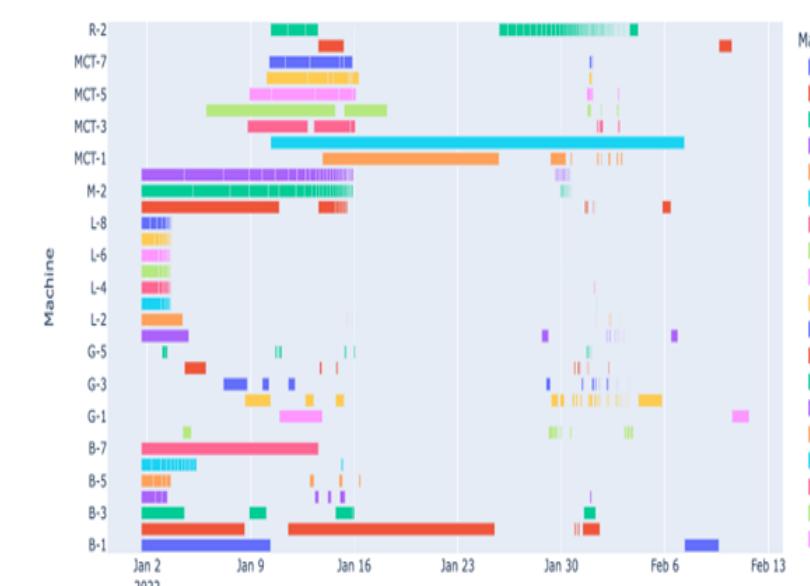
Optimal makespan  
443.93 (H)  
End time  
2022-01-20 02:40:13

Case 4

Start time

2022-01-01 14:44

- 간트 차트



- Optimal makespan, End time

Optimal makespan  
986.61 (H)  
End time  
2022-02-11 17:20:50



Case 3, Case 4의 경우 모두, (GA 알고리즘을 사용하였을 때의 Makespan)  $\leq$  (LPT rule을 사용했을 때의 Makespan)

## Conclusion

### 결론

이번 프로젝트를 진행한 효동기계공업 뿐만 아니라 생산품을 제조하는 대부분의 중소기업들은 주문받은 제품을 생산하기 위하여 생산 일정계획 및 납기 산정을 중요한 이슈로 다루고 있다. 따라 더욱더 효율적으로 생산 일정계획 및 납부기한을 맞추어야 하지만 여전히 현장 숙련자의 경험과 노하우에 의존하여 상황에 맞게 설비운영을 변경할 수밖에 없는 상황이 기계의 고장이나 작업 시간, 기계의 이용률등의 변동 사항들을 즉각 반영하기 어려운 현황이다. 이로 인해 현금 흐름은 물론 공정 개선에 많은 어려움을 보유하게 되어 회사의 경영 상태를 악화시키는 문제를 야기한다. 따라서 빠르게 공장내 상황을 반영할 수 있는 스케줄링 프로그램의 필요성이 강조되었다. 특히 Job Shop 생산 방식과 같은 경우는 Flow Shop 생산 방식에 비해 고려해야 할 요소가 많아 해당 프로그램의 중요도가 더욱 대두되는 상황이다. 이번 연구에서는 휴리스틱 방법인 유전 알고리즘을 이용하여 사용자가 입력한 값에 따라 생산 스케줄링을 간트차트로 시각화 해주는 프로그램을 제작하였으며, 이를 바탕으로 공장 내 변동 사항이 발생시 신속하고 가시성 있게 생산 계획을 도출 가능하게 하였다. 또한 기존 일정계획 알고리즘과 비교하여 해당 시스템의 효용성을 검증하였다. 해당 프로그램으로 생성된 생산 계획을 바탕으로 빠르게 기본안을 설정하고 현장 내 전문가의 의견을 반영하여 최종 생산계획안을 결정한다면 공장내/고객별 변동사항을 즉각 반영 가능할 것으로 기대된다.

### 기대 효과

#### 01 기존 생산 방식의 비효율성 개선

기존에는 생산 라인에서의 일정 계획에 대한 문제를 실무자의 경험으로만 계획 및 관리를 함으로 인해 공장 내 변동이 발생했을 때 이를 즉각 반영할 수 없었다. 이를 개선하기 위해 생산계획 모듈을 제작함으로써, 공장 내 변동 사항이 발생해도 이를 반영하여 유전 알고리즘을 이용해 근사해를 빠르게 찾아 새로운 생산계획을 도출해주는 프로그램을 제작하여 신속성과 가시성을 확보하였다.

#### 02 공장 내 CAPA를 반영한 생산계획 도출

납기 준수는 기업의 경쟁력을 결정짓는 중요 요소이다. 그러나 생산용량을 정확히 반영하지 못한 생산 계획은 초과근무 등의 부하 조정 방법을 추가적으로 이용해서 납기를 만족시켜야 하기에 비용을 발생시킨다. 본 프로젝트는 공장 내 설비 및 상황과 제작 부품별 Process time을 반영한 생산계획으로, flow time을 최소화하였기에 추가 발생 비용의 절감을 기대 가능하다.

#### 03 Job shop 생산 공정의 최적화

Flow Shop 생산 방식과 다르게 Job Shop 생산 방식의 경우 고려해야 하는 요소가 많아 매우 어려운 조합 최적화 문제로 분류된다. 따라서 모든 제약 조건을 사람이 수기로 반영하여 최적값을 도출하는 것은 시간, 비용적 문제로 불가능하다. 따라 휴리스틱 방법인 유전 알고리즘을 생산 계획에 적용하여 근사 최적해를 도출하였다는 점에서 의의가 있다. 또한 기존 생산계획 알고리즘인 SPT rule, LPT rule과 비교하였을 때 Flow time이 감소하였다는 사실의 확인을 통해 모듈 성능의 효과성을 검증하였다.

# Conclusion

## 한계

### 01 도메인 지식의 부족

본 프로젝트를 진행하기 위해 공장 내 다양한 요인을 고려하였지만, 회사 내 대외비등의 이유로 외부 공정의 고려 요소를 모두 반영하지 못하고 주요 요인만을 parameter로 설정하여 분석을 진행하였다는 한계가 있다. 이러한 한계를 극복하기 위해서는 실제 스케줄링 프로그램을 제작하는 과정에서 공장 내 물류 행태와 실제 가능 동원 인력, 공정 진행 시 고려 요소등의 다양한 도메인 지식을 반영해야한다.

### 02 실제 공장 데이터의 부재

본 프로젝트에서는 회사내 대외비등의 이유로 공장에서 실제로 누적된 RAW DATA가 아닌 논문 내에 기재되어있는 데이터를 기반으로 프로젝트를 수행하였다. 임의의 데이터가 아닌 실제 회사내 판매 제품을 기반으로 스케줄링을 진행했다는 점에서는 본 연구의 의의가 있지만, 실제로 해당 공장에 활용하기 위해서는 공장의 실제 주문 및 제조 현황 및 구체적인 공정 데이터를 기반으로 분석을 시행해 타당성을 검증해야한다.

### 03 목적 함수의 다양성 부재

본 연구에서는 목적 함수를 한가지로 설정하였다(FLOW TIME 최소화) 한계가 있다. 최대한 생산시간을 단축하여 납기를 맞추는 경우와, 납기만 맞추되 비용을 최소화는 경우등 공장 내 현황에 따라 목적 KPI는 바뀔 수 있는데, 이러한 점을 고려하지 못했다는 한계가 있다. 공정 기기별 가공비나 공장 별 가공비, 물류비등을 고려하거나, 주문자가 원하는 우선순위에 따라 목적함수를 설정하고 스케줄링을 진행할 수 있다면 더욱 효과적인 스케줄링이 될 것이라는 점을 인지해야한다.

### 04 GA(유전 알고리즘)의 한계

유전 알고리즘을 이용한 최적해는 문제 크기나 구조에 상관없이 최적 해에 근접한 해를 효과적으로 도출할 수 있다는 의의가 있지만, exact solution를 알 수 없다. 본 연구에서는 한계를 보완하기 위해 SPT, LPT rule 등을 이용하여 비교해서 성능을 검증했지만, 해당 결과 값이 최적이라고 평가할 수 없다는 한계가 있다. 따라서 도출된 해를 바탕으로 더 좋은 최적해를 만들기 위해서는 전문가 지식을 바탕으로 결과를 평가하고, 목적함수값의 패널티를 변경하면서 최적의 스케줄링 솔루션을 경험적으로 알아내야한다.

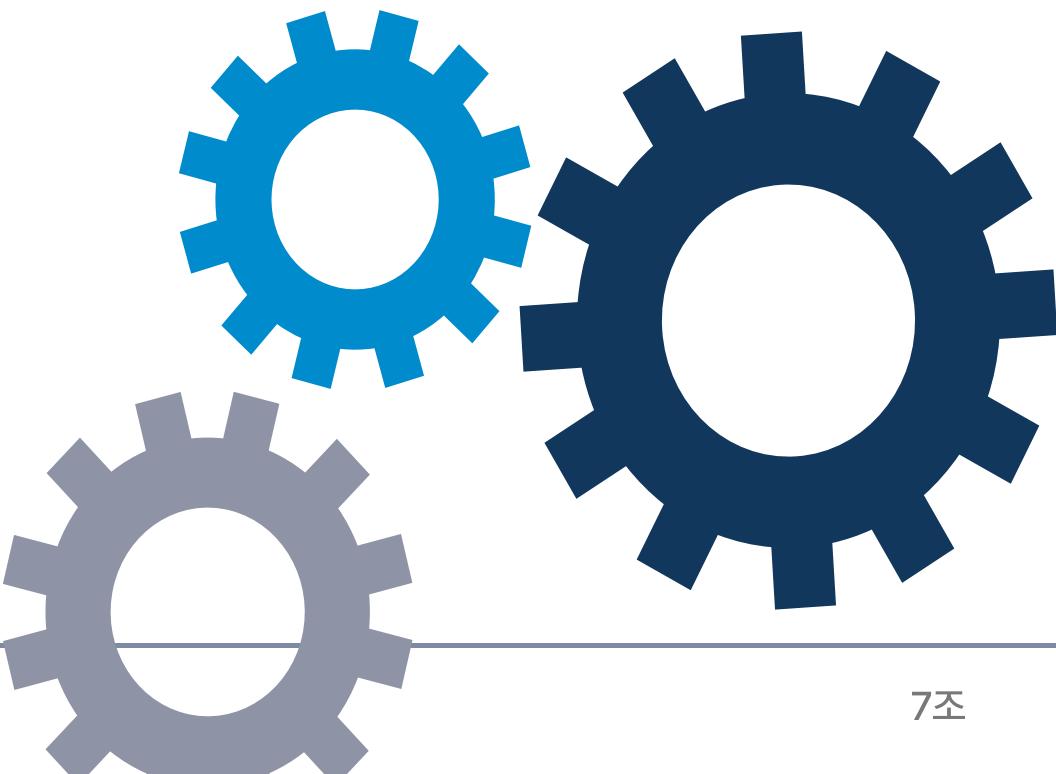
## References

### 논문

- 강인욱, 강호연, 이종연. (2019). Genetic Algorithm을 이용한 Job-Shop 공정의 생산 스케줄링 최적화 분석. 대한산업공학회 추계학술대회 논문집, (), 2681-2688.
- 박쌍균. "개별공정 형태의 자동차 부품 제조시스템에 대한 생산 스케줄링 사례연구." 국내석사학위논문 한양대학교 대학원, 2016. 서울
- Aquinaldo, S. L., Cucuk, N. R., & Yuniaristanto. (2021). Optimization in job shop scheduling problem using genetic algorithm (study case in furniture industry). IOP Conference Series.Materials Science and Engineering, 1072(1) doi:<https://doi.org/10.1088/1757-899X/1072/1/012019>
- Zhou Ermeng, Zhu Jin, Deng Ling. (2017) Flexible job-shop scheduling based on genetic algorithm and simulation validation. MATEC WEB OF CONFERENCES (JAN 2017), 2047 doi : <https://doi.org/10.1051/matecconf/201710002047>

### 기타 참고 자료

- [https://www.bing.com/images/search?view=detailV2&ccid=2U4Tg2c1&id=44F1F719BFF346DA75813A96282E1756620DC15C&thid=OIP.2U4Tg2c1mE\\_RTeW63YpjQQHaEZ&mediaurl=https%3a%2f%2fimages](https://www.bing.com/images/search?view=detailV2&ccid=2U4Tg2c1&id=44F1F719BFF346DA75813A96282E1756620DC15C&thid=OIP.2U4Tg2c1mE_RTeW63YpjQQHaEZ&mediaurl=https%3a%2f%2fimages)
- [https://file.irgo.co.kr/data/BOARD/ATTACH\\_PDF/4eb6bf431a9c29fc0bf33badb7ca5e8.pdf](https://file.irgo.co.kr/data/BOARD/ATTACH_PDF/4eb6bf431a9c29fc0bf33badb7ca5e8.pdf)
- IBK가 만드는 중소기업 CEO리포트 2017년 4월호
- CEO story(김동섭)\_20107년4월호(ibk경제연구소)
- 20200526효동기계공업 회사소개서 (최종본)
- 한국과학기술정보연구원, 대형 부품의 냉간 및 열간 단조
- <https://blog.naver.com/iim3headq/221128347064>
- <http://www.hyodongmachine.co.kr/?c=1/3>
- <https://www.kamp-ai.kr/front/dataset/AiData.jsp>



경영과학설계 - 7조

감사합니다