NS4S: Neighborhood Search for Scheduling Problems via Large Language Model (IJCAI-25)

26. 01. 30

황규상

I Introduction

Background:

❖ Job shop scheduling problem (JSP)

- ✓ 조합 최적화에서 가장 기본적이고 광범위하게 연구된 스케줄링 문제 중 하나로
  여러 개의 작업(job)이 각자 정해진 순서의 공정(operation)를 서로 다른 기계(machine)에서 처리해야 할 때,
  모든 제약을 만족하면서 전체 작업 완료 시간(makespan)을 최소화하도록 작업 순서를 결정하는 문제
- ✓ NP-hard 최적화 문제에 속함
  - 문제가 복잡하고 커질수록, 가능한 경우의 수가 '기하급수적으로' 늘어남
  - 현실적인 시간 안에 완벽한 최적해를 찾는 것이 매우 어려움
- ✓ 전통적으로 정확해법, 휴리스틱, 메타 휴리스틱 등을 사용

❖ Large Language Models (LLMs)

- ✓ LLM은 대규모 언어 모델(Large Language Model)의 약자로,
  텍스트 데이터를 학습하여 인간처럼 언어를 이해, 생성, 요약, 번역하는 인공지능(AI) 모델
  문맥을 파악하고 다음 단어를 예측하는 방식으로 작동하며, 챗봇, 콘텐츠 생성 등 다양한 자연어 처리 작업에 쓰임
  (ex: GPT, Gemini)

I Introduction

❖ 현재 LLMs을 이용한 heuristic search의 장점

- ✓ 강력한 추론 능력
  - 복잡한 스케줄링 제약조건과 상호작용 이해가능
- ✓ 문제의 패턴 학습, 식별에 탁월
  - 휴리스틱 탐색전략 개발에 강점
- ✓ 자연어 해석을 이용한 유연한 알고리즘 설계 가능

❖ 기존 모델의 한계

- ✓ 전체 알고리즘 생성방식이 지배적
  - JSP와 같은 복잡한 제약 조건 문제의 해의 품질 유지 실패 가능성 높음
- ✓ LLMS 고유의 hallucination 현상으로 인한 유효하지 않은 결과 생성 가능

❖ NS4S의 접근 방식

- ✓ LLMs-guided neighborhood search 프레임워크 제안
  - 전체알고리즘 생성 대신 local search에서의 neighborhood selection 가이드
  - 수동적인 파라미터 튜닝 없이도 메타 휴리스틱이나 확장된 문제에 적용가능
- ✓ Verification evolutionary 프레임워크 도입
  - Gradient정보와 유사한 피드백 제공으로 점진적인 휴리스틱 생성 개선

Ⅱ LLMs-guided Neighborhood Search

❖ Neighborhood Search

- ✓ 현재 해 주변에 있는 해들을 탐색하며 더 나은 해로 이동하는 탐색 방법
  - 초기 해가 존재하고 그 해에서 하나의 move로 생성된 후보 해들 중 최적해로 이동
- ✓ 본 논문에서는 Ding et al. [2019]의 논문에서 정의한 neighborhood structure를 사용
- ✓ Ding et al. [2019]에서 정의한 neighborhood structure(move의 형태) :
  - Same-machine insertion(N^{π}): operation 하나를 같은 머신 내 다른 위치로 삽입
  - Machine reassignment + insertion (N^{k}): operation을 다른 머신으로 옮긴 후 삽입

❖ Ding et al. (2019)에서 인용한 기존 search procedure

1. **초기해 생성 및 선택:**
   - 랜덤 초기 스케줄 생성, 이후 해들 중 makespan이 가장 작은 해 선택
2. **Critical path 도출 및 Critical operation 집합 추출**
   - Critical path = 경로 위의 작업 중 하나라도 지연되면 전체 스케줄의 완료 시간(makespan)이 증가하는 작업들의 연쇄
   - Critical path에 관여하는 operation들을 골라 집합으로 추출
   - $C = \{o \mid o \text{ is on critical path}\}$
3. **각 Critical operation에 대해 가능한 모든 move생성 및 평가**
   - 모든 $o \in C$ 에 대해 가능한 move를 생성하여 최소 makespan을 기준으로 최적의 move를 선택합니다

Ⅱ LLMs-guided Neighborhood Search

❖ LLMs-guided Neighborhood Search procedure

1. **초기해 생성 및 선택:**
   - 랜덤 초기 스케줄 생성, 이후 해들 중 makespan이 가장 작은 해 선택
2. **초기해 S의 모든 operation i에 대해 F값 계산 및 집합 생성(임의로 CF라 지칭)**
   - $F = \max\{R_i + p_i + Q_i + W(i)\}, \forall i \in \{O_1 \dots, O_k\}$ (2)
   - $W(i) = \text{CLIP}\left\{1 - \frac{t_i}{rand(N)}, 1\right) \times w_i, \forall i \in \{O_1 \dots, O_k\}$ (3)
3. **모든 o ∈ CF 에 대해 가능한 move를 생성**
4. **각 move에 대해 새 F 계산 이후 F값의 감소가 가장 큰 move를 선택**

R_i = i 의 가장 빠른 시작 시간
p_i =i의 작업 처리시간
Q_i =i의 후속 경로 중 종료까지의 가장 긴 경로의 길이

w_i = 작업 i에 대한 가중치
t_i =작업i의 마지막 이동 이후의 반복횟수
N =모든 작업(job)의 평균 작업(operation) 수

❖ FJSP 예제

✓ **문제 설정**

- Jobs
  - Job1 : J1 = O_1 → O_2 → O_3
  - Job2 : J2 = O_4 → O_5 → O_6
- Machines
  - M1,M2

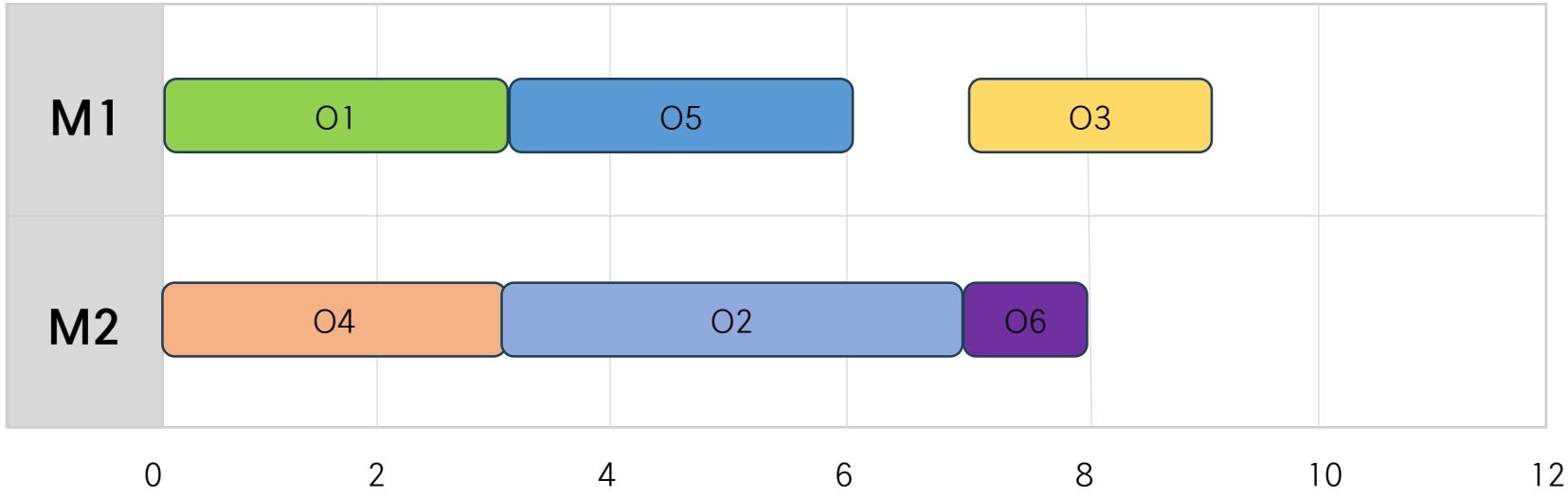
Ⅱ LLMs-guided Neighborhood Search

✓ Processing time

O_P	O_1	O_2	O_3	O_4	O_5	O_6
M1	3	2	2	2	3	4
M2	2	4	3	3	2	1

✓ 초기해 S

- M1 = O_1 → O_5 → O_3
- M2 = O_4 → O_2 → O_6

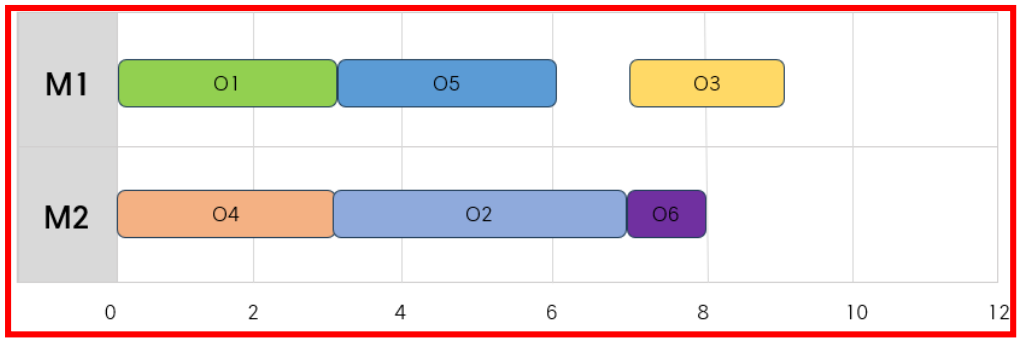


Ⅱ LLMs-guided Neighborhood Search

✓ **Processing time**

O_P	O_1	O_2	O_3	O_4	O_5	O_6
M1	3	2	2	2	3	4
M2	2	4	3	3	2	1

문제 상황



✓ **초기해 S**

- **M1** = $O_1 \rightarrow O_5 \rightarrow O_3$
- **M2** = $O_4 \rightarrow O_2 \rightarrow O_6$

❖ **Ding et al. (2019)에서 인용한 기존 tabu search procedure**

✓ 문제 상황속 Critical path:

- $O_1 \rightarrow O_2 \rightarrow O_3$
- $O_4 \rightarrow O_2 \rightarrow O_3$
- $C = \{o \mid O_1, O_2, O_3, O_4\}$

✓ 모든 $o \in C$ 에 대해 가능한 move를 생성하여 최소 makespan을 기준으로 최적의 move를 선택

- 예: O_1 을 이동시킬시 가능한 move

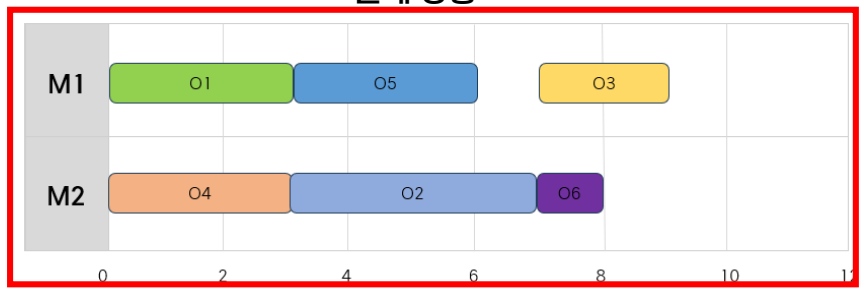
1. **M1** = $O_5 \rightarrow O_1 \rightarrow O_3$ **M2** = $O_4 \rightarrow O_2 \rightarrow O_6$
2. **M1** = $O_5 \rightarrow O_3$ **M2** = $O_1 \rightarrow O_4 \rightarrow O_2 \rightarrow O_6$
3. **M1** = $O_5 \rightarrow O_3$ **M2** = $O_4 \rightarrow O_1 \rightarrow O_2 \rightarrow O_6$

Ⅱ LLMs-guided Neighborhood Search

❖ LLMs-guided Neighborhood Search procedure

- ✓ 초기해 S의 $F = \max\{R_i + p_i + Q_i + W(i)\}, \forall i \in \{O_1 \dots, O_k\}$ 값 계산
- ✓ W(i)는 직관적으로 이해하기 쉽게 M1의 위 O면 W(i)=1, 아니면 0 으로 가정
- ✓ 예시) O_2 의 F를 구하는 과정:
  - R_2= 3(M2에서 O4 뒤 시작), p_2= 4, Q_2 = 2(O2이후 작업 O3, O6의 작업 중 긴 경로의 값) , $W(2)$ = 0 (M2의 작업)

문제 상황



O_p	R_i	p_i	Q_i	$W(i)$	$R_i + p_i + Q_i + W(i)$
O_1	0	3	6	1	**10**
O_2	3	4	2	0	9
O_3	7	2	0	1	**10**
O_4	0	3	6	0	9
O_5	3	3	2	1	9
O_6	7	1	0	0	8

- ✓ $CF = \{O_1, O_3\}$이므로 모든 $o \in CF$ 에 대해 가능한 move를 생성
- ✓ 각 move에 대해 새 F 계산 이후 F값의 감소가 가장 큰 move를 선택

Ⅱ LLMs-guided Neighborhood Search

❖ LLMs-guided 평가전략의 장점

- ✓ 지역 최적점(local optima) 탈출
  - LLMs가 생성한 휴리스틱을 기반으로 작업 가중치를 동적으로 조정함으로써 지역최적점 탈출
- ✓ 가중치를 통한 문제 특화 지식 학습
- ✓ 탐색의 다양성 유지

❖ The main framework of the NS4s algorithm

Algorithm 1 The main framework of the NS4S algorithm

Input: JSP Instance
Output: the best solution found so far S^*

$S^* \leftarrow S \leftarrow Init()$,
$f^* \leftarrow S.makespan$
while stopping condition is not met **do**
  $move(o^*, k) \leftarrow LLMGuidedNeighborEval(S)$ /*Section 3.2*/
  $S \longleftarrow S \oplus move(o^*, k)$
  $f \leftarrow S.makespan$
  $WeightUpdateByLLMs()$ /*Section 3.3*/
  if $f^* > f$ **then**
    $S^* \leftarrow S, f^* \leftarrow f$
  end if
end while
Return S^*

s^*= 전역 최적해 s= 현재해
f^*= 목적 함수값
$move(o^*, k)$= 작업 o^*를 위치 k로 재배치

Ⅲ VeEvo(verification-based evolutionary framework)

❖ VeEvo(verification-based evolutionary framework)

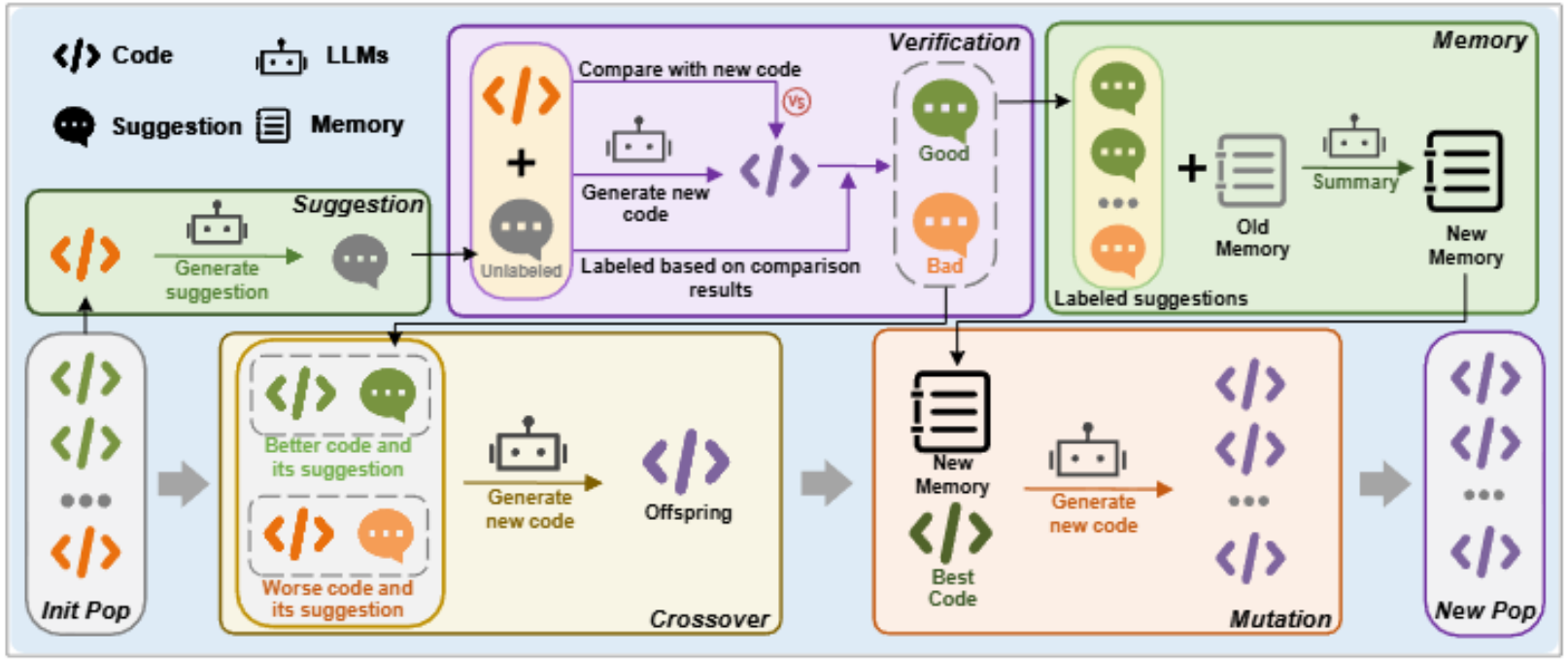


Figure 1: Schematic illustration of VeEvo.

❖ Initialization:

- ✓ 작업 설명, heuristic 정의 및 예시가 포함된 프롬프트를 기반으로 LLMs가 가중치를 업데이트하는 전략을 정의하는 초기 휴리스틱 코드를 생성

Ⅲ VeEvo(verification-based evolutionary framework)

❖ Suggestion Generation:

- ✓ LLMs를 통해 heuristic 코드를 순차적으로 처리하여 자연어 제안(suggestion)을 생성
  - 이는 진화적 crossover를 안내하는 gradient와 같은 정보 역할을 함
- ✓ LLMs의 불확실성 영향을 완화를 위해, 성능이 낮은 개체(fitness 기준 하위 50%)에 대한 제안은 " Unlabeled " 로 표시
- ✓ Unlabeled 제안은 추가 검증을 거침
- ✓ 예시 제안: "# Unlabeled Suggestion: [Unlabeled] Increase weightfor moved operations, decrease for unmoved with conditionsfavoring current optimization goal"
- ✓ 번역 : ("# 미분류 제안: [미분류] 현재의 최적화 목표를 고려하여, 이동이 발생한 작업에는 가중치를 높이고, 이동되지 않은 작업에는 가중치를 낮추는 방향을 제안한다.")

❖ Verification Process:

- ✓ 성능이 낮은 개체와 그에 관련된 제안은 LLMs에 의해 처리되어 개선된 heuristic을 생성
- ✓ 새로운 heuristic은 이전 버전과 비교 및 평가를 거침, 성능이 우수하면 "Good suggestion" 라벨이 지정, 성능이 열등하면 "Bad suggestion" 라벨이 지정
- ✓ 예시 제안: "#Good Suggestion: [High Quality] Increase weight for moved operations..."
- ✓ 이러한 품질 지표는 후속 연산에 정보 제공

Ⅲ VeEvo(verification-based evolutionary framework)

❖ Crossover Operation:

- ✓ LLMs는 부모 코드, 제안 및 해당 품질 라벨을 기반으로 자손(offspring) 코드를 생성
- ✓ 효과적이지 않은 제안을 폐기하는 대신 “bad suggestion” 레이블과 함께 유지
  - 학습 과정에 negative gradient 정보를 제공

❖ Memory Mechanism

- ✓ 제안의 성능을 지속적으로 평가·반영하여 진화하는 지식 베이스를 유지
- ✓ 예시: “Penalize stagnation and prioritize diverse solutions. Re-member, non-negative weights are crucial for algorithm stability”.
  번역 :(“탐색 정체(stagnation)를 패널티로 억제하고, 다양한 해를 우선하라.또한 알고리즘의 안정성을 위해 가중치는 항상 음수가 되지 않도록 유지해야 한다.)
- ✓ 새로운 제안은 기존 기억과 함께 품질 라벨을 통해 학습됨
- ✓ 성능이 낮은 제안은 유사한 비효율적 전략의 채택 확률을 감소시키는 데 활용

❖ Mutation Operation

- ✓ 프레임워크는 가장 성능이 좋은 개체와 축적된 메모리를 활용하여 LLMs가 다양한 변이체를 생성하도록 유도
- ✓ 이러한 새로운 개체들은 crossover로 생성된 자손들과 결합하여 다음 세대의 모집단을 형성

❖ VeEVo는 학습 과정에서 이러한 프로세스를 반복적으로 실행 후 테스트 단계에서 진화된 최적의 휴리스틱을 Algorithm 1 (Line 7)에 직접 통합

IV Experiment

❖ 테스트 문제

- ✓ JSP: 160개의 인스턴스로 구성된 TA 및 DMU 벤치마크 인스턴스를 활용
- ✓ FJSP: 169개의 다양한 인스턴스를 포함하는 BR, HU, 그리고 Vdata
- ✓ FJSP-SDST: 양한 복잡도를 가진 20개의 인스턴스가 포함된 SDST-Hudata 벤치마크 세트

❖ Baselines Algorithms:

- ✓ 우선순위 규칙(Priority Dispatch Rules, PDRs):
  - 우선순위 규칙은 JSP에서 처리 가능한 operation들 중 어떤 작업을 먼저 수행할지를 결정하는 간단한 휴리스틱
  - 초기해 생성에 사용
- ✓ 최첨단 메타휴리스틱(State-of-the-art Metaheuristics):
- ✓ 학습 기반 방법론(Learning-based Methods):
  - EYRL, WSRL, LKRL등 사용

❖ 실험 설정(Experimental Settings):

- ✓ VeEvo 프레임워크를 위한 Python과 neighborhood search 알고리즘을 위한 C++을 결합, 단일 CPU E5-2697v3에서 실행
- ✓ ReEvo[2024]와의 공정한 비교를 위해, VeEvo의 population size를 10으로, 최대 평가 횟수(maximum evaluations)를 100으로 설정
- ✓ JSP 및 FJSP에 대해서는 10초, FJSP-SDST에 대해서는 30초의 차단 시간(cutoff times)을 적용

IV Experiment

❖ 평가 지표

- ✓ $RPD = (C'_{max} - C^*_{max})/C^*_{max} \times 100\%$
- ✓ * 은 최적해를 의미

❖ JSP 의 TA 벤치마크 인스턴스 결과

Algorithm		Size								Avg
		15×15	20×15	20×20	30×15	30×20	50×15	50×20	100×20	
SPT	Obj	1546.1	1813.5	2067.0	2419.3	2619.1	3441.0	3570.8	6139.0	2952.0
	Gap	25.89%	32.82%	27.75%	35.27%	34.44%	24.11%	25.54%	14.41%	27.53%
FDD/WKR	Obj	1808.6	2054.0	2387.2	2590.8	3045.0	3736.3	4022.1	6620.7	3283.1
	Gap	47.15%	50.57%	47.61%	45.02%	56.30%	34.77%	41.50%	23.39%	43.29%
MWKR	Obj	1464.3	1683.6	1969.8	2214.8	2439.0	3240.0	3352.8	5812.2	2772.1
	Gap	19.15%	23.35%	21.81%	23.91%	25.17%	16.86%	17.95%	8.31%	19.56%
MOPNR	Obj	1481.3	1686.7	1968.3	2195.8	2433.6	3254.5	3346.9	5856.9	2778.0
	Gap	20.53%	23.55%	21.71%	22.83%	24.94%	17.37%	17.68%	9.15%	19.72%
L2D	Obj	1547.4	1774.7	2128.1	2378.8	2603.9	3393.8	3593.9	6097.6	2939.8
	Gap	25.96%	30.03%	31.61%	33.00%	33.62%	22.38%	26.51%	13.61%	27.09%
RASCL	Obj	1339.8	1509.3	1793.1	2038.1	2261.5	3030.8	3125.1	5578.9	2584.6
	Gap	9.02%	10.58%	10.87%	13.98%	16.09%	9.32%	9.89%	3.96%	10.46%
EYRL	Obj	1447.8	1645.8	1933.2	2189.1	2403.2	3217.3	3338.2	5845.0	2752.5
	Gap	17.85%	20.59%	19.53%	22.39%	23.32%	16.03%	17.41%	8.91%	18.25%
NS4S(GPT3.5)	Obj	1239.6	1389.5	1648.6	1820.0	2025.6	**2773.8**	2855.0	**5443.3**	2399.4
	Gap	0.87%	1.81%	1.94%	1.70%	3.98%	**0.00%**	0.39%	**1.45%**	1.52%
NS4S(GPT4)	Obj	**1239.0**	**1388.2**	**1647.1**	**1814.9**	**2012.6**	**2773.8**	**2849.2**	5446.2	**2396.4**
	Gap	**0.82%**	**1.70%**	**1.85%**	**1.41%**	**3.31%**	**0.00%**	**0.19%**	1.50%	**1.35%**

Table 1: Summary of results on the TA benchmark in JSP.

- ✓ RASCL이 TA 인스턴스에서 10.46% 라는 주목할 만한 평균 gap을 달성하여 다른 규칙 기반 휴리스틱 및 딥러닝 방법들을 능가
- ✓ 반면, NS4S는 이 성능을 능가하여 gap을 1.35%로 크게 줄임

IV Experiment

❖ DMU 벤치마크 인스턴스 결과

Algorithm		Size								Avg
		20×15	20×20	30×15	30×20	40×15	40×20	50×15	50×20	
SPT	Obj	4951.5	5690.5	6306.2	7036.0	7601.2	8538.1	8975.4	10132.8	7404.0
	Gap	64.13%	64.57%	62.57%	65.91%	55.88%	63.00%	50.37%	62.20%	61.08%
FDD/WKR	Obj	4666.3	5298.2	6016.5	6827.3	7420.0	8210.9	9150.2	9899.6	7186.1
	Gap	53.57%	52.52%	54.12%	60.09%	51.42%	55.52%	52.53%	57.26%	54.63%
MWKR	Obj	4909.9	5489.0	6252.9	6925.0	7484.2	8460.9	8906.0	9807.0	7279.4
	Gap	62.15%	58.16%	60.95%	63.16%	52.87%	61.11%	48.93%	56.40%	57.97%
MOPNR	Obj	4513.2	5052.3	5742.8	6491.9	7105.5	7870.7	8436.5	9408.0	6827.6
	Gap	49.16%	45.17%	47.14%	51.97%	44.72%	49.22%	40.79%	49.61%	47.22%
L2D	Obj	4215.3	4804.5	5557.9	5967.4	6663.9	7375.8	8179.4	8751.6	6439.5
	Gap	38.95%	37.74%	41.86%	39.48%	35.38%	39.38%	36.20%	38.86%	38.48%
RASCL	Obj	3610.0	4028.9	4522.0	5106.0	5731.9	6584.1	7242.1	7176.9	5500.2
	Gap	19.36%	15.98%	16.35%	20.00%	17.49%	25.42%	21.54%	14.66%	18.85%
EYRL	Obj	3839.7	4332.8	5012.5	5524.7	6108.8	6858.6	7479.1	8150.7	5913.4
	Gap	26.56%	23.55%	28.69%	29.21%	24.46%	30.06%	24.93%	29.51%	27.12%
NS4S(GPT3.5)	Obj	3117.3	3587.9	4077.9	4519.8	5120.1	5608.0	6313.4	6735.5	4885.0
	Gap	2.89%	3.17%	4.63%	5.86%	4.61%	6.36%	5.63%	7.09%	5.03%
NS4S(GPT4)	Obj	**3106.9**	**3579.5**	**4047.9**	**4499.0**	**5099.2**	**5553.2**	**6288.3**	**6721.5**	**4861.9**
	Gap	**2.57%**	**2.96%**	**3.91%**	**5.37%**	**4.21%**	**5.33%**	**5.23%**	**6.84%**	**4.55%**

Table 2: Summary of results on the DMU benchmark in JSP.

- ✓ GPT-4 기반 모델은 4.55%의 평균 gap을 달성하여, RASCL의 18.85%를 실질적으로 능가
- ✓ 초기 프롬프트에 명시적인 가중치 업데이트 방향 지침 없이 달성됨
  -> VeEvo의 능력을 강조

IV Experiment

Algorithm	Barnes		Brandimarte		Dauzere		Hurink-rdata		Hurink-edata		Hurink-vdata	
	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)
IJA	2.40%	21.03	8.50%	15.43	6.10%	180.8	3.90%	19.72	4.60%	15.24	2.70%	17.85
RegGA	–	–	8.39%	280.1	–	–	–	–	–	–	3.20%	191.4
2SGA	–	–	3.17%	57.6	–	–	–	–	–	–	0.39%	51.43
SLGA	–	–	6.21%	283.28	–	–	–	–	–	–	–	–
FIFO+EET	27.91%	0.019	28.98%	0.017	15.00%	0.036	17.38%	0.018	19.89%	0.016	7.14%	0.019
MWKR+EET	54.71%	0.018	39.50%	0.019	24.69%	0.036	26.60%	0.018	44.68%	0.018	8.96%	0.02
MOPNR+EET	50.66%	0.017	43.39%	0.018	32.17%	0.038	26.47%	0.019	43.67%	0.018	13.14%	0.019
EYRL	13.83%	0.391	13.24%	0.406	11.02%	0.808	12.09%	0.275	15.54%	0.271	5.37%	0.272
WSRL	17.88%	1.516	30.04%	1.305	8.88%	2.716	11.02%	1.421	16.66%	1.424	4.41%	1.405
LKRL	28.96%	2.048	13.59%	2.054	15.68%	3.917	16.51%	1.591	23.01%	1.558	6.96%	1.544
NS4S(GPT3.5)	1.26%	1.016	0.17%	1.027	0.66%	3.513	0.14%	1.704	0.07%	1.074	0.10%	1.154
NS4S(GPT4)	**1.20%**	**2.026**	**0.05%**	**1.455**	**0.31%**	**5.117**	**0.00%**	**2.412**	**0.01%**	**1.448**	**0.09%**	**2.609**

Table 3: Summary of results on the FJSP benchmarks.

❖ FJSP

- ✓ NS4S는 EYRL, WSRL, LKRL과 같은 강화 학습 접근 방식보다 솔루션 품질 우수
- ✓ 대등한 계산 효율성을 유지
- ✓ GPT-4 모델은 모든 벤치마크 세트에서 우수한 솔루션 품질을 보여주었으나, 계산 시간은 다소 증가

❖ FJSP-SDST

- ✓ 복잡한 제약 조건을 처리하는 본 연구 방식의 강건함 (robustness)을 강조
- ✓ NS4S는 솔루션 품질과 계산 효율성 모두에서 Memetic Algorithm (MA)을 능가
- ✓ 벤치마크 인스턴스 전반에 걸려 9개의 새로운 상한선을 수립

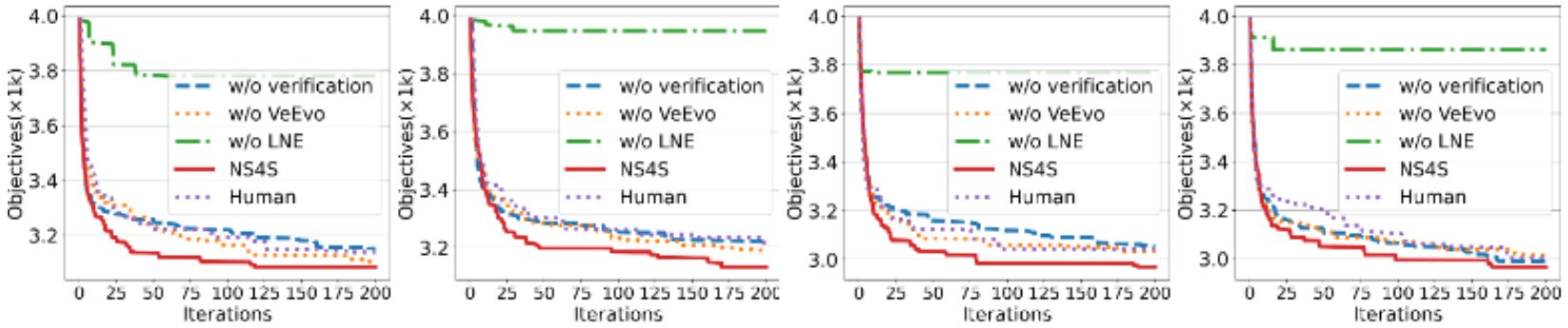
Ins.	LB	IFS	GA		TS		MA			NS4S(GPT3.5)			NS4S(GPT4)		
		Best	Best	Avg	Best	Avg	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)
la01	609	726	801	817	721	724	721	724	6	**721**	721.0	2.16	**721**	721.5	9.87
la02	655	749	847	870	737	738	737	737	7	**737**	737.5	7.16	**737**	737.5	9.62
la03	550	652	760	789	652	652	652	652	7	**652**	652.0	0.40	**652**	652.0	0.37
la04	568	673	770	790	673	678	673	675	9	**673**	673.0	1.04	**673**	673.0	1.71
la05	503	603	679	685	602	602	602	602	8	**602**	602.0	1.76	**602**	602.0	0.84
la06	833	950	1147	1165	956	961	953	957	12	**945***	946.2	10.66	**945***	946.5	9.84
la07	762	916	1123	1150	912	917	905	911	18	**902***	905.6	9.10	904	907.9	12.92
la08	845	948	1167	1186	940	951	940	941	15	**940**	942.4	14.32	**940**	941.4	13.02
la09	878	1002	1183	1210	1002	1007	989	995	22	**984***	988.2	9.19	**984***	986.0	14.36
la10	866	977	1127	1156	956	960	956	956	29	**953***	955.4	3.39	956	956.0	0.61
la11	1087	1256	1577	1600	1265	1273	1244	1254	33	1236	1239.9	9.32	**1232***	1235.6	8.93
la12	960	1082	1365	1406	1105	1119	1098	1107	26	**1070***	1081.2	12.52	1077	1081.9	8.11
la13	1053	1215	1473	1513	1210	1223	1205	1212	24	1180	1182.6	15.73	**1172***	1177.6	18.29
la14	1123	1285	1549	1561	1367	1277	1257	1263	27	1237	1244.6	12.15	**1234***	1242.4	15.49
la15	1111	1291	1649	1718	1284	1297	1275	1282	29	1261	1266.9	10.49	**1258***	1262.2	17.12
la16	892	1007	1256	1269	1007	1007	1007	1007	12	**1007**	1007.0	0.29	**1007**	1007.0	0.46
la17	707	858	1007	1059	851	851	851	851	12	**851**	851.0	2.79	**851**	854.0	2.37
la18	842	985	1146	1184	985	988	985	992	10	**985**	991.4	4.21	**985**	987.4	14.33
la19	796	956	1166	1197	951	955	951	951	16	**951**	953.0	11.81	**951**	951.5	9.07
la20	857	997	1194	1228	997	997	997	997	12	**997**	997.0	1.01	**997**	997.0	1.37

Table 4: Summary of results on SDST-HUdata benchmark in FJSP-SDST.

IV Experiment

❖ Analysis

- ✓ 알고리즘 구성 요소의 중요성을 평가하기 위해, SVW 벤치마크에서 가장 도전적인 4개의 인스턴스를 대상으로 소거 연구(ablation studies)를 수행
- ✓ 검증 메커니즘이 없는 버전(w/o verification), LLMs 유도 neighborhood 평가가 없는 버전(w/o LNE), VeEvo 프레임워크를 ReEvo [Ye et al., 2024]로 대체한 버전, 그리고 전문가가 설계한 가중치 조정 휴리스틱 버전(Human version)



(a) SWV12 (b) SWV13 (c) SWV14 (d) SWV15

Figure 2: Evolution of the makespan by NS4S and others on four hardest instances in *SWV* benchmark.

- ✓ 전체 NS4S 구현체가 단축된 계산 시간으로 더 우수한 해를 얻을 수 있다는 점은 이러한 구성 요소들을 통합함으로써 발생하는 시너지 효과를 입증

V Conclusion

❖ Methodological Contributions

- ✓ 첫째, 복잡한 해 공간을 통해 탐색 과정을 효과적으로 안내하기 위해 동적 가중치 조정을 사용하는 LLMs-guided neighborhood evaluation 전략을 도입
- ✓ 둘째, 생성된 휴리스틱에 대한 엄격한 검증을 통해 LLMs의 hallucination 영향을 체계적으로 완화하는 verification-based evolutionary framework (VeEvo)를 개발
- ✓ 셋째, LLMs가 생성한 가중치 조정 전략의 통합이 어떻게 탐색 과정을 해 공간의 유망한 영역으로 효과적으로 유도할 수 있는지 입증

❖ Key Results

- ✓ Taillard의 인스턴스에서 평균 최적성 갭(optimality gap)을 10.46%에서 1.35%로 크게 감소
- ✓ FJSP의 맥락에서는 Brandimarte의 인스턴스에서 평균 최적성 갭을 13.24%에서 0.05%로 더욱 낮춤
- ✓ 가장 주목할만한 점은, FJSP-SDST에 대해 NS4S의 접근 방식이 9개의 새로운 상한선(upper bounds)을 수립