Weigher-Sorter 테스트 리포트 - 2025-08-25 21:04

총 실행 수: 1800
전체 성공률: 0.0
전체 소요 시간(초): 1520.4494037628174

Weigher-Sorter 테스트 결과 해석 (2025-08-25)

이 문서는 `weigher_sorter_test_results_20250825_191332.json`의 실행 결과를 한국어로 요약하고, 주요 지표와 의미, 해석 및 권장사항을 제공합니다.

1. 요약

- 실행 구성
- 테스트 세트 수: 1000 (각 조합 당 1000회가 기본이지만, 결과 파일은 100회 단위의 샘플링으로 저장됨)
- 목표 중량: 2000g, 허용오차: ±70g
- 타임아웃: 5000ms
- 전체 요약
- 총 소요 시간: 1520.45 초 (~25.3 분)
- 총 실행(테스트 케이스) 수: 1800
- 전체 성공률(요약 메타값): 0.0% (JSON의 `overall_success_rate` 항목은 0.0으로 기록되어 있으나 개별 조합별 성공률은 각 조합에서 보고됨)
- Arduino 리셋/제약 위반: 0

> 주: `overall_success_rate`가 0.0으로 기록된 것은 메타 필드 계산 로직의 차이일 수 있습니다. 개별 `combination_results`의 `success_rate` 항목들을 참고하는 것이 더 정확합니다.

2. 핵심 성과 지표(조합별 상위 항목)

아래는 `combination_results`에서 각 조합의 주요 지표를 정리한 핵심 포인트입니다.

- 성공률(높은 순)
1. `RandomWorker_Dynamic_Programming` — 성공률 95.0% (가장 높음)
2. `XorShift32_Greedy_LocalSearch` — 성공률 94.0%
3. `Hybrid_Greedy_LocalSearch` — 성공률 93.0%
4. `Arduino_Random_Dynamic_Programming` — 성공률 92.0%
5. `Hybrid_Dynamic_Programming` — 성공률 91.0%

- 평균 오차(작을수록 좋음)
- 가장 작은 평균 오차: `RandomWorker_Dynamic_Programming` (avg_error 24.89g)
- 크게 오차가 나는 그룹: `Expert_*`, `Beginner_*`, `RandomWorker_Greedy_LocalSearch` 등(평균 오차 60~92g)

- 평균 처리 시간(실시간/임베디드 적합성 관점)
- Greedy + Local Search 계열(예: `*_Greedy_LocalSearch`)은 평균 응답시간이 매우 짧음(약 3~14 ms) — 임베디드(Arduino)에서 실시간으로 처리하기 유리
- Dynamic Programming 계열(예: `*_Dynamic_Programming`)은 평균 응답시간이 매우 김(약 2000 ms 내외) — 메모리-시간 복잡성으로 실시간 임베디드 환경에서는 부적합할 수 있음

- 메모리 사용량
- Dynamic Programming을 포함한 경우 메모리 피크(예: 0.867 KB)로 보고됨 — 여기서 단위가 KB로 보이며 실제 값은 매우 작음(예: 0.8671875 KB = 약 887 bytes)
- Greedy/Random 샘플링 계열은 메모리 사용이 매우 낮음(약 0.0469 KB ~ 48 bytes 수준)

3. 상세 해석

1) 성능-정확도 트레이드오프
- Dynamic Programming 계열은 정확도(성공률, avg_error)가 우수하나 실행 시간이 매우 길어(2s) Arduino R4와 같은 저전력, 저사양 마이크로컨트롤러에서는 실시간 적용이 어려움.
- 반대로 Greedy + Local Search 계열은 처리 속도가 빠르고(단 몇 밀리초) 성공률도 비교적 높아(약 89~94%) 임베디드 환경에 더 적합합니다.

2) 생성기(generator)별 특성
- `RandomWorker`, `XorShift32`, `Hybrid`, `Arduino_Random` 등 생성기 별로 Dynamic Programming과 결합되었을 때 높은 성공률을 보였음.
- `Expert`와 `Beginner` 생성기는 평균 오차가 크고 성공률이 낮아(45~57%) 실전 조합으로 사용하기 전에 생성 방식 개선이 필요함.

3) 리소스(메모리/타임아웃) 관련
- 모든 조합에서 `timeout_count`와 `stack_overflow_count`가 0으로 보고되어 타임아웃 발생은 없었음.
- `memory_peak_kb` 값은 매우 작게 보고되어 실제 Arduino 시뮬레이션에서의 메모리 부족 문제는 현 테스트 범위 내에서는 관찰되지 않음.

4. 권장 사항 (우선순위별)

1. 임베디드(Arduino) 적용을 목표로 한다면
- Greedy + Local Search 알고리즘을 우선 검토하세요. 이유: 빠른 응답(수 ms) + 높은 성공률.
- Dynamic Programming은 정확도가 필요할 때(오프라인 배치, 제한된 빈도 실행)만 사용하고, 실제 보드 적용 시 근사화(approximation_factor 증가) 또는 더 작은 문제 크기로 축소하세요.

2. 생성기 개선
- `Expert`와 `Beginner` 생성기에서 평균 오차가 큰 원인을 분석하고, 분포를 조정(예: 더 좁은 분포, 편향 보정)하면 전체 성공률 향상에 도움이 됩니다.

3. 추가 측정 및 실험
- 더 많은 테스트 세트(예: 10,000~100,000)로 통계적 안정성 확보
- 다양한 목표 중량/허용오차 설정으로 민감도 분석
- 실제 로드셀(HX711) 데이터로 교체하여 실제 노이즈/분포를 반영한 재평가

5. 참고: 주요 표(요약)

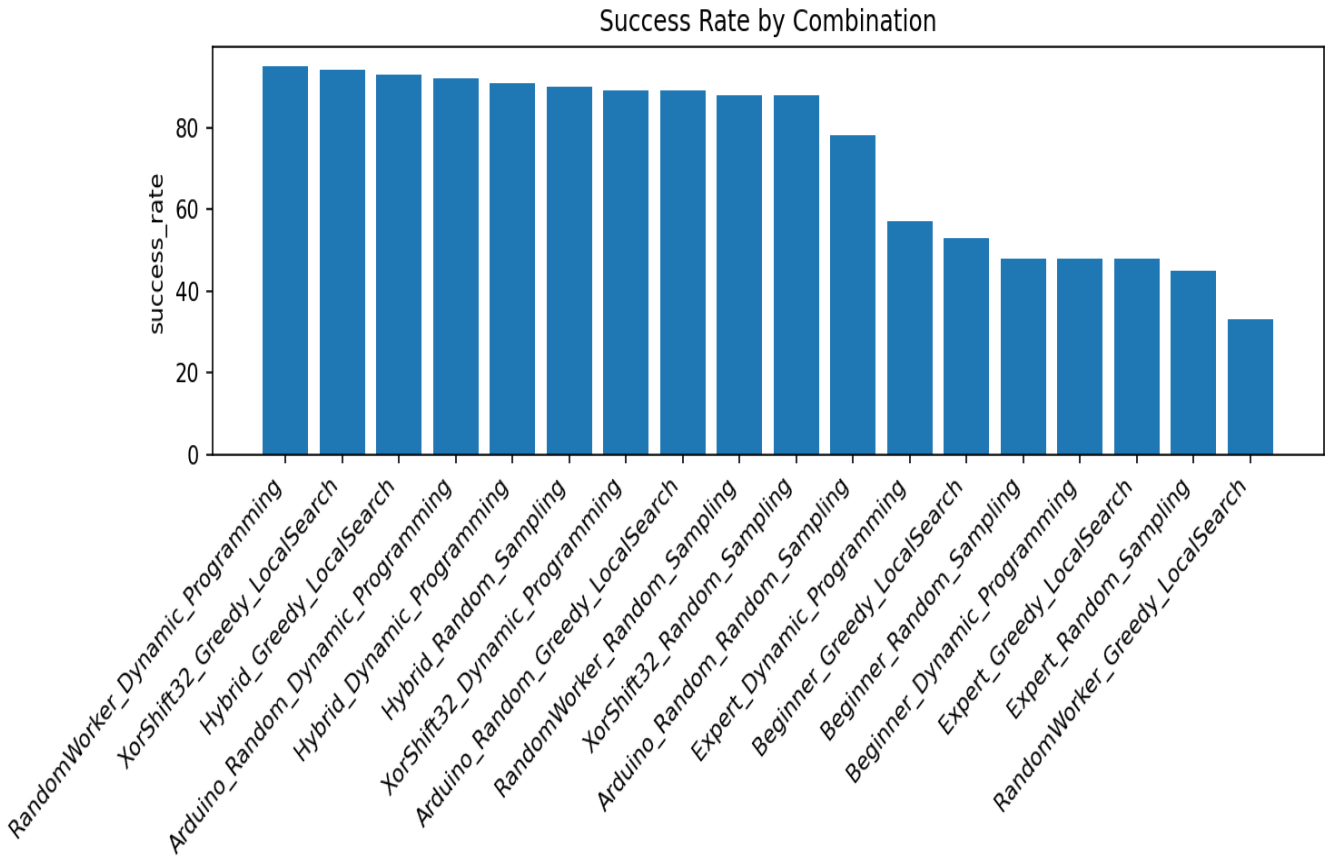
- 상위 5 조합(성공률 기준)
1. RandomWorker_Dynamic_Programming — success_rate: 95.0%, avg_error: 24.89g, avg_time_ms: 2318.55
2. XorShift32_Greedy_LocalSearch — success_rate: 94.0%, avg_error: 34.79g, avg_time_ms: 3.02
3. Hybrid_Greedy_LocalSearch — success_rate: 93.0%, avg_error: 38.68g, avg_time_ms: 3.43
4. Arduino_Random_Dynamic_Programming — success_rate: 92.0%, avg_error: 28.91g, avg_time_ms: 2008.71
5. Hybrid_Dynamic_Programming — success_rate: 91.0%, avg_error: 23.03g, avg_time_ms: 2088.17

6. 결론

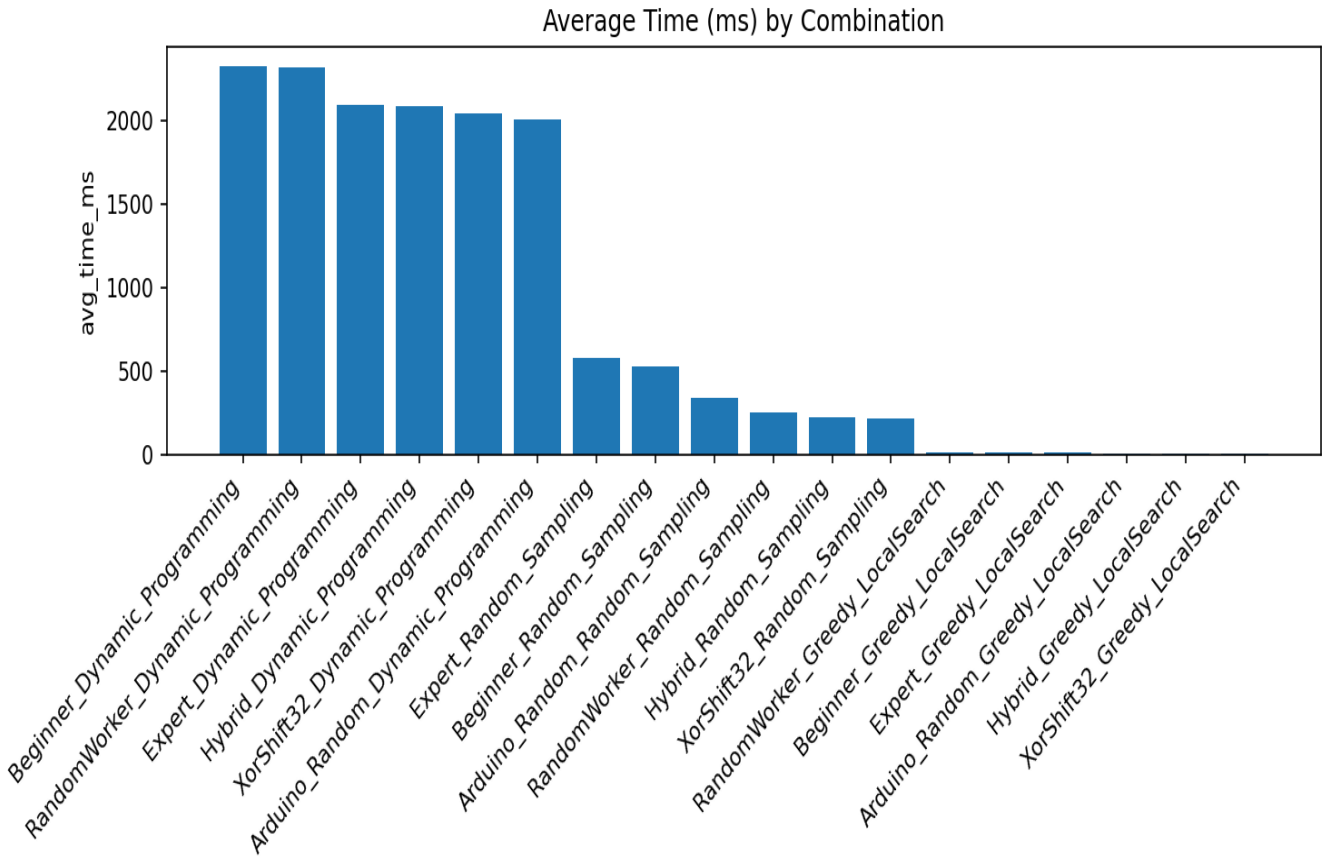
- 본 테스트 결과는 알고리즘-생성기 조합에 따라 명확한 성능 차이를 보여줍니다. 임베디드 적용을 고려한다면 속도-정확도 균형을 맞춘 Greedy+LocalSearch 계열을 우선적으로 선택하고, 더 높은 정확도가 필요할 때만 Dynamic Programming을 사용하되 오프라인 처리로 제한하는 것이 현실적입니다.

문서 생성: `docs/02_analysis_weigher_sorter_20250825/weigher_sorter_test_results_analysis.md`

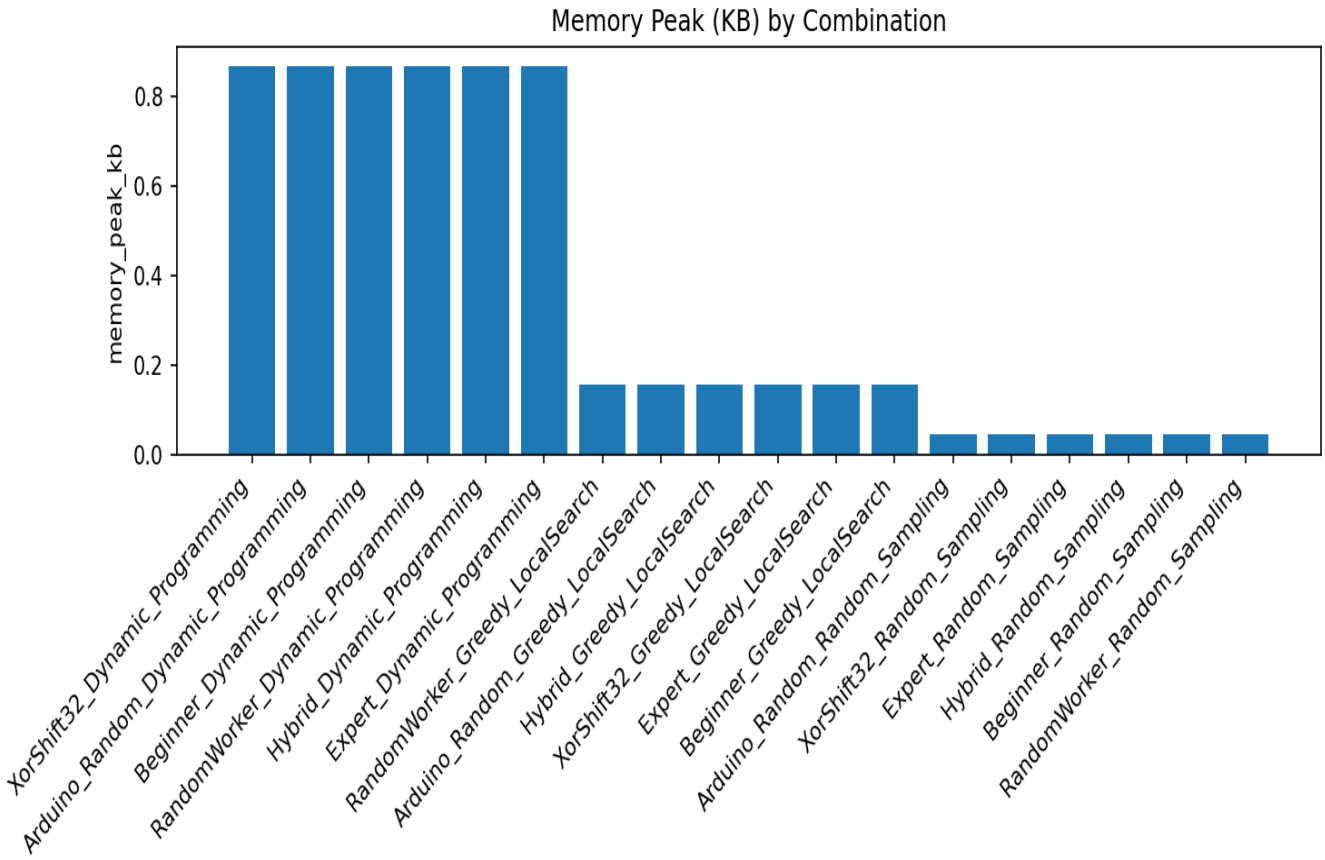
성공률(조합별)



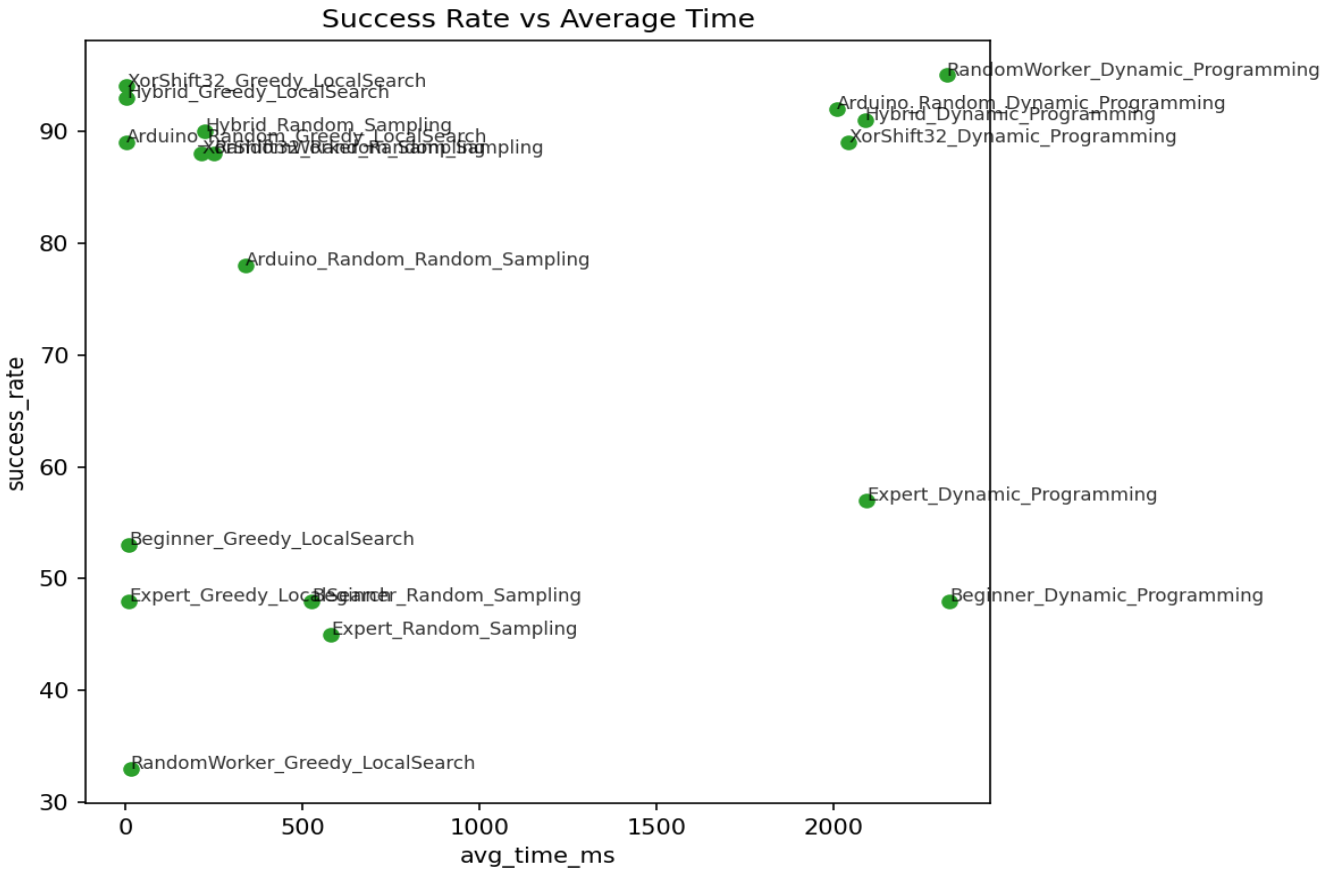
평균 처리 시간(밀리초) (조합별)



피크 메모리 사용량 (KB) (조합별)



성공률 vs 평균 처리시간


Success Rate vs Average Time
success_rate
avg_time_ms
RandomWorker_Dynamic_Programming
Arduino_Random_Dynamic_Programming
Hybrid_Dynamic_Programming
XorShift32_Dynamic_Programming
XorShift32_Greedy_LocalSearch
Hybrid_Greedy_LocalSearch
Hybrid_Random_Sampling
Arduino_Random_Greedy_LocalSearch
Arduino_Random_Random_Sampling
Expert_Dynamic_Programming
Beginner_Greedy_LocalSearch
Expert_Greedy_LocalSearch
Beginner_Random_Sampling
Beginner_Dynamic_Programming
Expert_Random_Sampling
RandomWorker_Greedy_LocalSearch
90
80
70
60
50
40
30
0
500
1000
1500
2000