

Stepwise Composition of Heuristic Components for Offline Strip Packing: A Controlled Study of Batch Reinforcement Learning

Wenru Lin

linwenru@mju.edu.cn

Minjiang University

Zhouwang Yang

University of Science and Technology of China

Lei Liu

Minjiang University

Research Article

Keywords: strip packing, constructive heuristics, reinforcement learning, fitted Q iteration, hyper-heuristics, algorithm portfolios

Posted Date: October 12th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11317441/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Stepwise Composition of Heuristic Components for Offline Strip Packing: A Controlled Study of Batch Reinforcement Learning

Wenru Lin^{1,2*}, Zhouwang Yang³ and Lei Liu²

¹School of Artificial Intelligence and Data Science, University of Science
and Technology of China, Hefei, China.

^{2*}School of Computer and Big Data, Minjiang University, Fuzhou, China.

³School of Mathematical Sciences, University of Science and Technology
of China, Hefei, China.

*Corresponding author(s). E-mail(s): linwenru@mju.edu.cn;

Abstract

Learning to select heuristic components during construction offers flexibility, but its value beyond static diversity is unclear. We study batch reinforcement learning for stepwise composition in offline two-dimensional strip packing. A controller selects item-selection/placement-rule pairs in a shared skyline engine via tree-based fitted Q iteration and a dense reward aligned with height before postprocessing. Evaluation uses family holdout on 224 development instances and 160 new same-generator test instances. On the 104-instance classical subset, adding the learned constructor to TWBF-18 and BBF-288 reduces mean gap by 0.305/0.163 points (paired 95% intervals exclude zero). Gains persist across four training seeds and the final set. The raw learned constructor trails the best train-selected fixed configuration. Development-set controls support state-dependent decisions; exploratory final-set controls do not confirm an advantage over an open-loop stage policy. A 12-configuration train-selected static portfolio reaches lower mean gap at about one tenth of the augmented BBF-288 portfolio’s CPU cost. At its CPU budget, random-swap local search outperforms the constructor and both augmented portfolios. Capping native threads cuts measured inference CPU by 94.6%. The findings establish modest portfolio complementarity and its limits against static portfolios and budgeted search.

Keywords: strip packing, constructive heuristics, reinforcement learning, fitted Q iteration, hyper-heuristics, algorithm portfolios

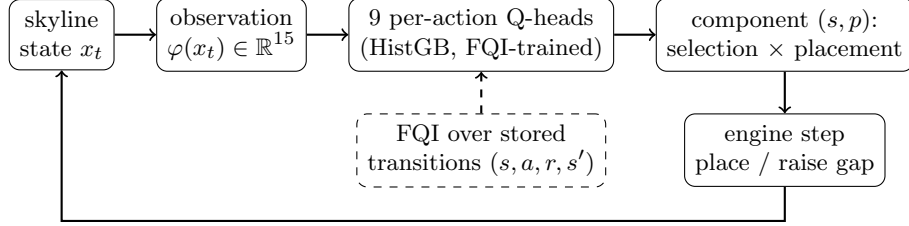


Fig. 1 One construction step of the learned composer. The reward is the waste-identity increment (Proposition 1); tower removal is a fixed postprocess.

1 Introduction

Two-dimensional strip packing (2DSP)—pack a set of rectangles into a fixed-width, open-topped strip so as to minimise the used height—is a classical NP-hard problem with a steady stream of heuristic literature. Constructive skyline heuristics [2] are the cheapest family: a single pass places items one by one. Two decades of variants—selection rules, placement policies, orderings—produced a robust finding that no variant dominates [22]. Two published shells exploit this diversity statically: BBF enumerates 288 strategy combinations [1] and TWBF 18 ordering $\times$ policy combinations [26], keeping the best packing per instance.

We ask whether a *learned* controller can compose the same components adaptively *within* one pass, choosing the selection rule and placement policy at each step as a function of the current partial layout. Two observations motivate the question: enumeration pays for diversity with redundant constructions, and the choice that matters varies within an instance, not only across instances.

Because “does learning help here” is easily confounded—adding *any* further constructor to a shell can only improve its per-instance best—we pre-registered a control battery that separates state-dependent learning from switching diversity, open-loop switching, and train-selected fixed configuration, plus an equal-CPU-budget comparison against random restarts and local search. All protocols, gates, and design revisions were fixed in dated design documents before the corresponding runs.

Findings

(i) *Standalone*, the learned constructor (raw) does not beat the best single fixed configuration; with a safety fallback (a two-member ensemble) it ranks fourth among the published single methods and fifth once the train-selected portfolio joins the comparison set (Table 2). (ii) *Composed* into the published TWBF-18 / BBF-288 shells, the learned policy adds significant value, and the augmented BBF-288 shell (4.405%) is numerically below FH (4.501%), the strongest published single method measured here—a difference that is not significant on a paired comparison (+0.095 [−0.385, +0.633]); a train-selected static 12-configuration portfolio from the same component space is lower still (4.238%). (iii) *Controls*: the composition gain significantly exceeds random stepwise switching, a stateless frequency-matched policy, and a fixed open-loop stage table on both shells (all six paired 95% CIs exclude zero)—on the development set the gain is consistent with state-dependent closed-loop decisions and

not explained by switching diversity, marginal action frequencies, or open-loop stage structure; on the untouched final set the open-loop stage table caught up with the learned member (Section 5.7). A train-selected fixed configuration arm is statistically indistinguishable from the learned member (better on TWBF-18, worse on BBF-288; indistinguishable does not imply equivalent). (iv) *Cost*: the learned member is never cost-competitive. Plain random-swap local search with not-worse acceptance given the same CPU budget as the learned member’s inference reaches 1.873% mean gap—below the augmented shell’s 4.405%; at 0.2s/instance it reaches 2.963%, below every constructive method and shell tested. (v) *Inference tax*: 94.6% of the learned policy’s measured raw CPU time disappeared when native thread caps were applied ($2160 \rightarrow 116$ s per 104 instances; per-call overhead $488\times$), consistent with per-row **predict** calls spawning OpenMP thread teams—a measurement artefact, not model computation. Timing claims in learning-based packing papers that report construction counts but not thread-controlled CPU time can be distorted by such artefacts.

Contributions

(1) The first offline, stepwise, batch-RL composer over named classic components for strip packing, with a terminal-consistent telescoping reward (closest prior art selects whole heuristics with GA in bin packing, or acts online; Section 2). (2) A pre-registered control battery testing whether the composition gain is specific to state-dependent, closed-loop decisions—on the development set the evidence is consistent with that attribution; on the final set the closed-loop-over-open-loop part is not confirmed—the paper’s central evidence. (3) An equal-CPU-budget accounting showing that budgeted random-swap local search dominates bbf-288, FH, and both learning-augmented variants on both axes at every budget tested (only bf-TN, ish-core, twbf-18, and the train-selected portfolio remain cheaper-and-worse frontier points), with an inference-tax artefact analysis (thread-control churn) that we believe should accompany all learning-based packing cost reports.

2 Related work

Constructive strip packing and the BF family

Burke et al. [2] introduced the skyline best-fit (BF) heuristic; successors added mechanisms and variants: vertical niches (BBF; 1), three-way policies and orderings (TWBF; 26), fitness-number selection and a random local search shell (ISH; 29). BBF and TWBF embed enumeration shells—run all 288 (resp. 18) configurations, keep the best. Our action vocabulary is exactly this published vocabulary, and the published shells are our composition partners. Level/shelf families [19] and the skyline-tabu framework of Wei et al. [28] populate the same unified protocol we reuse. Within the constructive line, the recent backtracking heuristic of Li et al. [15] (BHA) strengthens large-scale results but, by its own height-sum comparisons, does not dominate FH on the classical C/N/CX sets; its cross-method table also mixes five hardware/time-limit configurations, the comparability gap our controlled protocol avoids. Further out on the quality axis, the skyline-guided bounded tree search of Li et al. [16] improves best-reported solutions for 21 of 112 benchmark instances of the rotation-allowed problem (its 112

include the 104 studied here); with no public implementation, it cannot be re-measured under our protocol, and that search-budgeted quality tier is not what our controlled comparisons contest.

Hyper-heuristics

Selection hyper-heuristics choose among low-level heuristics (LLHs) during search [3]. Closest to us: Terashima-Marín et al. [24] and López-Camacho et al. [17] evolve condition-action rule sets with genetic algorithms that select BL-family heuristics during construction for *bin packing*; Garrido and Riff [7] evolve per-instance open-loop schedules of constructive heuristics for 2DSP; Wang et al. [27] apply Q-learning to select *perturbation* operators within a population framework for irregular packing; Kallestad et al. [11] learn operator selection *inside ALNS* (an improvement framework over routing and scheduling domains—not online packing rule selection). We differ on four axes: the problem (offline rectangular strip packing), the action set (factorised component micro-actions rather than whole heuristics), the learner (batch value-function RL with a terminal-consistent telescoping reward rather than GA or tabular Q), and the evaluation (family holdout with shell baselines and a control battery).

Learning-based construction for packing

Offline neural constructors learn new policies end-to-end: pointer-network orderings [5], actor-critic placement [13], learned scoring functions [10, 32], GNN agents [30], and MCTS self-play [12]. None selects *among named classic components* stepwise; all learn new decision machinery. Online DRL hyper-heuristics [25, 31] select BF placement rules in the online list model (irrevocable placement, unknown future) and report gains around one percent; our setting is offline, with replay and terminal-consistent credit assignment. Per-instance *algorithm selection* for packing is supervised and one-shot [18, 20, 21]; our decision granularity is per step. Notably, Kaleta and Śliwiński [10]—whose setting additionally allows rotation—include a “best of ten variants” oracle baseline that their neural method fails to beat, the same diversity-wall phenomenon we document systematically.

Batch reinforcement learning

FQI [4] and neural fitted Q [23] learn Q-functions from fixed transition sets with supervised regressors; batch RL has learned dispatching rules in job-shop scheduling [6] and orienteering [14]. To our knowledge, tree-ensemble FQI has not been applied to component selection inside constructive combinatorial heuristics.

3 Method

3.1 Problem, rotation convention, and engine

Figure 1 shows one construction step of the learned composer (state, observation, per-action Q-heads, component, engine).

An instance gives n rectangles ($w_i \times h_i$) and a strip width W ; the goal is a feasible, non-overlapping placement of minimal height H . **Orientation convention.** We follow

Table 1 Observation features (15), all online-computable.

group	features
skyline shape	segment count; lowest-gap width; min/mean/SD/max height
remaining items	count ratio; area ratio; mean/max width mean height; slender share
progress	cumulative buried area; step index; last height increment

the BF literature’s convention [2]: items are width-normalised in preprocessing ($w_i \geq h_i$ after a fixed 90-degree relabelling), and the tower-removal postprocess (TR) may rotate top vertical items during reinsertion. Hence the solved variant is the *rotation-normalised* one, not the strict fixed-orientation one; all methods compared share this convention. Under it, $\text{LB} = \max(\lceil \sum_i w_i h_i / W \rceil, \max_i h_i)$ remains a valid lower bound: each item’s placed height is at least its normalised (shorter) side.

The engine is a deterministic skyline constructor: each step takes the lowest gap (leftmost tie), selects an item by a configurable rule, places it at a gap edge by a configurable policy, and raises unfillable gaps (the area beneath is buried). TR is applied once at the end of each construction for all arms; it is a cheap postprocess, not an action. All packings pass a geometric validator (completeness, orientation, bounds, non-overlap, height consistency).

3.2 MDP formulation

- **Environment state** x_t : the full partial layout and remaining item set. The policy reads a 15-dimensional *observation* $\varphi(x_t)$ (Table 1) computed online; as with any compressed observation, distinct environment states can share it (a POMDP reading), which does not affect the validity of the training procedure but bounds the theoretical claims we make.
- **Actions** (9): item-selection rule $\in \{\text{widest-fit, first-fit, fitness-number}\} \times \text{placement policy} \in \{\text{LM, TN, SN}\}$. Precise semantics (including tie handling and first-fit’s list order) follow the cited source papers and are documented in the replication package.
- **Reward**: let $\Phi_t = W \cdot H_t - A_t$, where H_t is the current strip height and A_t the area placed so far ($\Phi_0 = 0$), and set $r_t = -(\Phi_{t+1} - \Phi_t)/(W \cdot \text{LB})$.
- **Horizon**: n steps; deterministic transitions.

Throughout, $g(H) = (H - \text{LB})/\text{LB}$ and the reported gap is $100 g(H)$ in percent.

Proposition 1 (telescoping reward). $\sum_{t=0}^{n-1} r_t = -g(H_n) - (\text{LB} - A/W)/\text{LB}$: *the return equals minus the terminal construction gap minus an instance-dependent constant (the second term is constant per instance).*

Proof. $\sum_t (\Phi_{t+1} - \Phi_t) = \Phi_n - \Phi_0 = WH_n - A$; divide by $W \cdot \text{LB}$ and rearrange; the last term is constant per instance. $\square$

Thus the dense reward tracks the terminal objective with no per-step misalignment; we do *not* claim this removes long-horizon credit assignment difficulty for the function approximator. Scope notes: the reward telescopes to the *pre-TR* construction height, i.e. “terminal-consistent” is scoped to the construction objective (TR is

Algorithm 1 Training and deployment (one fold)

Require: training instances $\mathcal{I}$; behaviour episodes $E=8$; rounds $K=5$; reuse episodes $E'=2$; $\varepsilon = 0.2$

- 1: collect behaviour episodes on $\mathcal{I}$ (4 random, 2 BF-like, 2 S2-like)
- 2: $\hat{Q}_0 \leftarrow$ per-action regressors fit on r
- 3: **for** $k = 1$ **to** $K - 1$ **do**
- 4: fit per-action regressors on $y_t = r_t + (1 - d_t) \max_{a'} \hat{Q}_{k-1}(\varphi(x_{t+1}), a')$ with $\hat{Q}_{k-1}$ **frozen** within the round
- 5: **end for**
- 6: collect E' ε -greedy episodes against $\hat{Q}_{K-1}$; refit K rounds on the union
- 7: **deploy:** greedy argmax policy; per instance, if its verified gap exceeds the train-fold best fixed configuration's, fall back to the fixed configuration (rate reported)

a shared postprocess whose gain is policy-dependent, not identical across arms: measured TR deltas are -2.21pp on the strongest fixed configuration vs -1.54pp on the learned policy's layouts); and the observation is a compressed summary (POMDP), so we report behaviour-level evidence rather than representation-level guarantees.

3.3 Learning: fitted Q iteration

Behaviour episodes per training instance (8): four uniform-random, two with a fixed BF-like rule (widest-fit/LM), two with a fixed S2-like rule (fitness-number/SN). From the pooled transition set, FQI fits one histogram gradient-boosting regressor per action ($\gamma = 1$, five rounds; per-fit subsample 10^5 rows with seeds derived per round $\times$ action; the full set is replayed each round, so no target network is needed). A second data pass adds two ε -greedy episodes ($\varepsilon = 0.2$) per instance against the frozen $\hat{Q}$ of the first pass and refits (Algorithm 1); terminal states are masked from the bootstrap. Training-budget note: the reuse pass is applied to the main arm and the Double-DQN arm (three reuse rounds there); the feature, macro-action, and wide-action ablation arms received no reuse pass, so arm-to-arm comparisons carry that asymmetry (recorded). Five rounds balance fit stability against the documented divergence risk of approximate FQI; the choice and its sensitivity are recorded in the design log.

3.4 Safety fallback

The deployed artefact pairs the learned policy with a per-instance safety fallback onto the train-fold best fixed configuration. The fallback member is selected on training folds only (no test information). We report raw, fixed, and deployed (min-of-two) numbers separately throughout; the deployed ensemble is a two-member combination and is described as such.

3.5 Function-class control

FQI's tree ensembles match the small hand-designed observation and the batch setting, train in minutes on CPU, and are deterministic. As a function-class falsification arm, a full Double-DQN + Dueling variant (MLP $15 \rightarrow 128 \rightarrow 128$, $\gamma = 1$, Huber loss, periodic

target sync, GPU training) was run under the same evaluation protocol (three reuse rounds; Section 5.6).

4 Experimental setup

Instances and splits

Development set: C1–C7 (21) [9], BKW/N (13) [2], NT (70) [8], PV (120 prospective instances: four geometry groups $\times$ two scales, $W=1000$); training additionally uses 800 self-generated instances from the PV generator (disjoint seed range 46–145). A 160-instance final set (seed range 146–165) was generated at the same time and *left untouched until the protocol was frozen* (the frozen design log is dated and included in the replication package); Section 5.7 reports it. Family holdout: train on all families except the held one (the PV fold additionally holds out generator slice 126–145). The PV fold is therefore *within-generator*: its training data includes the same generator’s slices 46–125, so PV results do not test cross-distribution generalisation; the final set is likewise a new-sample, *same-generator* test (seeds 146–165), not a cross-distribution one. Per-fold split: C/BKW/NT/PV folds hold out 21/13/70/120 instances and train on the remaining 1003/1011/954/744 behaviour-episode instances (base plus generated, counts computed from the actual split); the final set is evaluated once with the pipeline retrained on all development instances including the 800 generated ones.

Baselines

Published methods in the same unified protocol: BF (LM/TN/SN), ISH-core, TWBF-18, BBF-288, FH. A fold-fair greedy-12 portfolio from the 864-configuration component space is reported as an analysis reference (fold means C/BKW/NT/PV: 3.18/1.57/5.05/4.80%). Control arms (Section 5.3): random stepwise switching, stateless frequency-matched sampling, fixed open-loop stage table, train-selected fixed configuration, equal-budget random restarts, and equal-budget random-swap search. The train-selected arm is the train-fold best single fixed configuration—one extra construction per instance, selected on training folds only; a conventional strong constructor, not a learned selector.

Search-arm specification

The budgeted search is a random-swap local search with not-worse acceptance: the core is the strongest single constructive configuration in our component space (perimeter ordering, fitness-number selection, shortest-neighbour placement, TR on); the initial sequence is perimeter-descending (sorting happens only at initialisation, never per candidate); the neighbourhood is the uniform random pair swap; the walk continues from an accepted (not-worse) sequence without restart; TR runs inside every candidate evaluation; the run terminates early at the area lower bound (18/104 instances in the budget arm; 9–25/104 on the curve depending on budget), and otherwise exhausts the budget, which is checked before each candidate evaluation (overshoot $\leq$ one pack); the initial construction, TR, and validation all count toward the budget; curve axes use measured actual CPU, not nominal budgets. The random-restart arm randomises the component action per step (not the item order). Learning and search arms share

Table 2 Standalone results (mean gap %). RL raw = learned policy + TR; fixed = train-fold best fixed configuration; deployed = per-instance min of the two (fallback ensemble). Published comparators in the same protocol. PV: shells not measured; fallback rate 0.86 analysed in text.

fold	RL raw	fixed	deployed	bf-TN	ish-core	twbf-18	bbf-288	fh	greedy-12
C (21)	6.93	6.84	4.54	7.63	4.62	4.72	3.22	2.48	3.18
BKW (13)	5.55	3.69	2.54	9.00	3.69	3.12	2.11	1.39	1.57
NT (70)	8.62	8.04	6.70	7.94	7.69	5.93	5.43	5.69	5.05
weighted	7.90	7.26	5.74	8.01	6.57	5.33	4.57	4.50	4.24

the same engine, orientation normalisation, and TR postprocess, so the comparison isolates the decision mechanism, not the machinery.

Statistics and timing

Instance-level paired bootstrap, 10,000 resamples, fixed seed; 95% percentile intervals; all CIs in this paper come from the same bootstrap script so that repeated quantities carry identical intervals. Timing: serial `process_time` (CPU seconds; note that `process_time` accumulates CPU across a process’s threads, so we cap native threads at one—OMP/BLAS/vecLib—and record wall time as secondary), three reps, median; measurement window on an otherwise idle machine; hardware/software: Apple M5, 16 GB RAM, Python 3.13, scikit-learn 1.5. A pipeline-level reproducibility note: an early draft of the pipeline admitted scheduler-dependent jitter of ± 0.3 points on fold means (insertion order of reuse episodes); the released pipeline fixes the ordering. All reported solutions and statistical outputs regenerate bitwise from a single code version; timing figures are medians of three repeated serial runs and are not claimed to be bitwise-reproducible.

5 Results

5.1 Standalone: the raw policy does not beat the best fixed configuration

Table 2 corrects the natural misreading: the raw learned constructor (7.90%) trails the best single fixed configuration (7.26%) and the ISH core; the fourth-place figure (5.74%) belongs to the two-member fallback ensemble, not to the learned policy alone. On the PV family the learned policy falls back on 86% of instances (raw 8.42%, deployed 5.96%). Because the PV fold’s training data still includes same-generator slices (Section 4), this is a *within-generator* weakness, not a distribution-shift effect; we record it as an open observation rather than explain it.

5.2 Composition: significant complementarity to published shells

Tables 3–4 give the composition results. The complementarity is broad-based (every family, both shells). **BBF-288 + RL reaches 4.405%, numerically below FH’s 4.501%**; the paired difference is +0.095 points with 95% CI $[-0.385, +0.633]$ (28/104

Table 3 The learned policy as one more member of the published shells (per-instance best; RL member = raw policy + TR). Gain = shell minus combination; 95% CI = paired bootstrap (10,000, fixed seed).

shell	n	shell alone	shell + RL	gain [95% CI]	wins
TWBF-18	104	5.333	5.028	+0.305 [+0.160, +0.470]	22/104
BBF-288	104	4.568	4.405	+0.163 [+0.062, +0.292]	12/104

Table 4 Composition per family.

shell	family	shell alone	shell + RL	gain [95% CI]
TWBF-18	C	4.722	4.365	+0.357 [+0.040, +0.873]
TWBF-18	BKW	3.115	2.484	+0.631 [+0.103, +1.333]
TWBF-18	NT	5.929	5.700	+0.229 [+0.093, +0.393]
BBF-288	C	3.221	2.963	+0.258 [+0.000, +0.754]
BBF-288	BKW	2.111	1.957	+0.154 [+0.000, +0.462]
BBF-288	NT	5.429	5.293	+0.136 [+0.057, +0.236]

Table 5 Control arms composed into TWBF-18 (mean gap %). Arms: random stepwise switching (5 seeds); stateless sampling from the RL policy’s marginal action distribution (5 seeds); fixed open-loop stage table (modal RL action per progress bucket, learned on training instances); train-selected fixed configuration; train-fold best fixed configuration (one extra construction per instance). Δ gain–RL: paired difference of gains, control minus RL (negative = RL better).

arm	alone	gain [95% CI]	Δ gain–RL [95% CI]
RL policy	7.90	+0.305 [+0.160, +0.470]	—
rand-step	8.51	+0.057 [+0.006, +0.144]	−0.248 [−0.436, −0.077]
freq-step	8.08	+0.115 [+0.064, +0.173]	−0.190 [−0.360, −0.044]
stage-table	9.74	+0.062 [+0.000, +0.144]	−0.242 [−0.435, −0.073]
train-sel-fixed	7.26	+0.520 [+0.289, +0.797]	+0.215 [−0.060, +0.518]

wins)—lower in mean, *not* significant, and we do not claim otherwise. The mechanism reading—TWBF enumerates static orderings, BBF strategy vectors, the learned policy adapts within the pass—is tested, not assumed, in Sections 5.3 and 5.8.

5.3 Controls: the gain is consistent with state dependence, but the learned member is not the strongest possible complement

Tables 5–6 and Figure 2 are the paper’s central evidence. The learned policy’s composition gain significantly exceeds (i) random stepwise switching, (ii) a stateless policy with the same marginal action frequencies, and (iii) a fixed open-loop stage table—on both shells, all six paired CIs excluding zero. These six comparisons were pre-registered; intervals are unadjusted per-comparison 95% CIs, and the BBF stage-table interval (upper bound −0.003) is marginal and read as such. The evidence is *consistent with*

Table 6 Control arms composed into BBF-288 (columns as in Table 5).

arm	alone	gain [95% CI]	$\Delta\text{gain-RL}$ [95% CI]
RL policy	7.90	+0.163 [+0.062, +0.292]	—
rand-step	8.51	+0.016 [+0.003, +0.034]	-0.146 [-0.278, -0.045]
freq-step	8.08	+0.023 [+0.004, +0.050]	-0.139 [-0.272, -0.039]
stage-table	9.74	+0.043 [+0.000, +0.101]	-0.119 [-0.259, -0.003]
train-sel-fixed	7.26	+0.119 [+0.043, +0.215]	-0.044 [-0.196, +0.093]

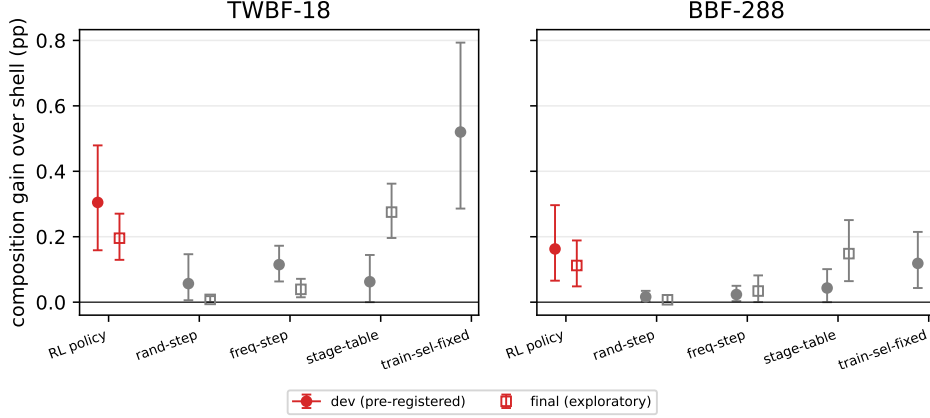


Fig. 2 Composition gain per arm over each shell (pp, paired 95% CI). Filled circles: development set (pre-registered comparisons of Tables 5–6); open squares: final set (exploratory, Table 14). The learned policy (red) clears every switching control on the development set while the train-selected fixed configuration bounds it from above; on the final set the open-loop stage table crosses over on TWBF-18 (Section 5.3).

state-dependent, closed-loop decisions driving the complementarity and *not explained by* switching diversity, marginal action frequencies, or this open-loop compression; we do not claim state dependence is the only possible mechanism, nor that no open-loop policy could match it (our stage table is a simple modal compression, not an optimised open-loop policy). The bounding fact is equally important: a train-selected fixed configuration is statistically indistinguishable from the learned member (better point estimate on TWBF-18, worse on BBF-288)—an absence of detected difference, not a demonstration of equivalence. The learned policy’s distinctive niche is the *strongest* shell: where another fixed config adds least, its within-pass adaptation still adds value.

5.4 Cost: budgeted local search dominates at every budget tested

Tables 7–8 give the decisive cost verdict. Handing the learned member’s own budget to plain random restarts already reaches 2.735%; handing it to random-swap local search reaches 1.873%—below FH (4.501%), below the augmented shell (4.405%),

Table 7 Equal-CPU-budget controls. Each instance’s budget equals the RL member’s measured CPU time (median of three; thread-clean measurement below). Arms: repeated uniform-random constructions (best-of); first-improvement single-swap local search over the sequence (strongest single constructive core, TR on). CPU caps are process-time, so the comparison is load-fair.

arm	alone	packs/inst	combo TWBF	combo BBF
RL policy	7.897	—	5.028 (+0.305)	4.405 (+0.163)
budget-rand	2.735	~800	2.735 (+2.598)	2.717 (+1.851)
budget-ls	1.873	~5400	1.873 (+3.461)	1.873 (+2.696)

Table 8 Budgeted local-search quality–time curve (mean gap % over the 104 instances; budget in CPU seconds per instance, process-time capped).

budget/instance	0.2 s	1 s	5 s	20 s	60 s
mean gap	2.963	2.307	1.928	1.598	1.390

Table 9 Cost–quality summary (104 instances; gap = weighted mean %; CPU = sum of serial median-of-3 process times; per-instance means in parentheses). Frontier points in bold. See also Figure 3.

method	gap	CPU sum (s)	CPU/instance (s)
bf-TN	8.006	0.47	0.005
ish-core	6.568	1.70	0.016
twbf-18	5.333	12.11	0.116
greedy-12 portfolio	4.238	19.4	0.186
bbf-288	4.568	91.30	0.878
fh	4.501	2829.97	27.2
rl-policy (artefact)	7.897	2159.69	20.8
rl-policy (thread-capped)	7.897	116.0	1.115
bbf-288 + RL (thread-capped)	4.405	207.3	1.99
LS@0.2s	2.963	22.3	0.214
LS@1s	2.307	95.7	0.920
LS@5s	1.928	448.5	4.313

below the component space’s per-instance oracle (3.42%). Even at 0.2 s/instance—one tenth of the augmented shell’s thread-clean cost—the search arm reaches 2.963%. **At no tested budget does the learned or composed constructor pay operationally.** We therefore withdraw any deployment recommendation for the learned member as implemented; the per-budget recommendation that our data supports is: millisecond $\rightarrow$ bf-TN; ~ 10 ms $\rightarrow$ ish-core; ~ 0.1 s $\rightarrow$ twbf-18; ≥ 0.2 s $\rightarrow$ budgeted local search; FH and BBF-288 appear nowhere on the frontier (Table 9).

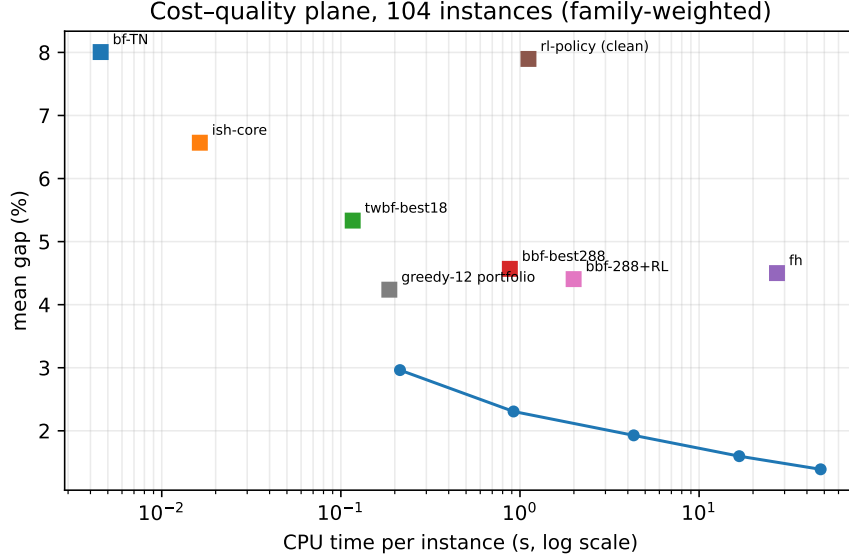


Fig. 3 Cost-quality plane on the 104 instances (family-weighted mean gap vs per-instance CPU time, log scale). The budgeted local-search curve dominates (better on both axes) bbf-288, FH, the raw learned policy, and the learning-augmented shell already at 0.2 s/instance; bf-TN, ish-core, twbf-18, and the train-selected 12-configuration portfolio remain cheaper-and-worse frontier points.

5.5 The inference tax: an OpenMP artefact

The learned policy’s striking raw cost (2160 s per 104 instances) turned out to be an artefact: per-row `predict` calls on nine tree models in a thread-uncontrolled process cost ~ 430 ms per construction step, vs 0.88 ms with native threads capped at one—a $488\times$ per-call overhead consistent with OpenMP thread-team churn (we verify the effect by removal, not by tracing the runtime internals). Profiling on five representative instances attributes 72–98% of rollout time to it (replication package, `off3-profile.csv`); the cap removes 94.6% of the member’s total CPU time ($2159.69 \rightarrow 116.0$ s; wall times and thread settings in the package). Thread-capped inference brings the RL member to 116 s (1.1 s/instance) and the augmented shell to 207 s. On this thread-capped accounting the shell is cheaper and lower-gap than FH (207 s, 4.405% vs 2830 s, 4.501%); the quality difference itself is not significant (paired 95% CI $[-0.385, +0.633]$), and none of it rescues the operational case against budgeted local search (Section 5.4).

The methodological lesson: construction counts and thread-uncontrolled CPU times systematically distort the cost accounting of learned packing policies; we recommend reporting thread-capped CPU time alongside.

5.6 Ablations and function class

Table 10: under the current training budgets, no design dimension improves consistently. Mechanism features help only on NT; the macro- and wide-action configurations

Table 10 Ablations (deployed mean gap % per held-out fold, canonical rerun).
Exploratory configuration comparisons: the reuse pass was applied to the main and Double-DQN arms only, so cross-arm differences confound the design change with the training budget.

arm	C	BKW	NT	PV	reading
main (9 actions, 15 feats)	4.54	2.54	6.70	5.96	best on C/BKW
+ mechanism features (19)	5.11	3.05	6.28	5.97	better on NT only
macro-actions ($m=5$)	5.24	3.61	7.39	5.96	higher gaps here
wide actions (36)	5.99	—	6.97	—	higher gaps here
Double-DQN + Dueling	5.22	2.99	7.44	5.82	no consistent edge

Table 11 Training-seed robustness (deployed mean gap % per fold; seed 42 = canonical; three further full-pipeline seeds).

fold	seed 42	seed 43	seed 44	seed 45	mean	sample sd
C	4.54	5.58	5.19	5.69	5.25	0.52
BKW	2.54	3.36	3.15	2.68	2.93	0.39
NT	6.70	6.94	6.84	6.84	6.83	0.10
PV	5.96	5.94	5.89	5.89	5.92	0.04

Table 12 Per-seed shell-composition gains (paired 95% CI): every training seed’s gain over both published shells is positive and significant.

seed	RL raw	TWBF-18 gain [CI]	BBF-288 gain [CI]
42 (canonical)	7.90	+0.305 [+0.160, +0.470]	+0.163 [+0.062, +0.292]
43	8.41	+0.245 [+0.116, +0.413]	+0.048 [+0.014, +0.091]
44	8.58	+0.221 [+0.129, +0.327]	+0.106 [+0.038, +0.188]
45	8.66	+0.181 [+0.093, +0.285]	+0.106 [+0.043, +0.178]

perform worse here; the Double-DQN arm is worse on three folds and marginally better on PV. Given the training-budget asymmetry noted in the caption, these are exploratory configuration comparisons, not causal attributions to the design dimensions. In the tested configurations, the neural function class showed no consistent advantage over tree-ensemble FQI—a scoped statement about these configurations, not about the method family.

5.7 Robustness: training seeds and the untouched final set

Two further robustness checks close the protocol: (i) three additional training seeds over the four folds, separating training randomness from sampling uncertainty (Table 11); (ii) one evaluation of the frozen pipeline on the 160 untouched final instances (Table 13).

Training randomness moves fold means by up to 1.15 points of range on C (sample sd 0.52) and 0.82 on BKW (sd 0.39), with the canonical seed at the favourable end, and is negligible on NT/PV (sd < 0.1); the fallback ensemble ranks fourth for all four

Table 13 Untouched final set (160 self-generated instances, seed range disjoint from all training and tuning): mean gap %. RL raw has no fallback in this table (the all-dev best-fixed fallback is reported in the replication package). Combo gains with paired 95% CI.

RL raw	bf-TN	twbf-18	bbf-288	LS@5s
8.00	6.89	6.37	5.75	3.66
shell		shell + RL	gain [95% CI] (wins)	
TWBF-18		6.18	+0.196 [+0.130, +0.271] (41/160)	
BBF-288		5.64	+0.112 [+0.050, +0.190] (18/160)	

Table 14 Final set, exploratory: control arms composed into the shells (gain over the shell alone, paired 95% CI; and paired gain difference vs the RL member). Added after the final set was opened.

arm	alone	TWBF gain [CI]	BBF gain [CI]
RL policy	8.00	+0.196 [+0.130, +0.271]	+0.112 [+0.050, +0.190]
rand-step	8.32	+0.008 [+0.000, +0.019]	+0.007 [+0.000, +0.022]
freq-step	7.99	+0.039 [+0.015, +0.071]	+0.034 [+0.000, +0.081]
stage-table	7.35	+0.275 [+0.195, +0.364]	+0.148 [+0.065, +0.255]
paired gain difference, control – RL:			
rand-step		−0.187 [−0.259, −0.124]	−0.105 [−0.185, −0.041]
freq-step		−0.156 [−0.231, −0.084]	−0.078 [−0.167, +0.002]
stage-table		+0.080 [+0.001, +0.162]	+0.036 [−0.056, +0.131]

seeds among the published single methods (fifth including the portfolio), while the composition gains remain positive and significant for every seed (Table 12). On the untouched final set the picture replicates qualitatively (Table 13): the raw policy again trails fixed rules (the all-dev best-fixed fallback reaches 5.92%, deployed 5.80%, fallback rate 0.88), shell complementarity is again significant in the same direction (41 and 18 wins of 160), and budgeted search again dominates everything constructive (3.66% at 5s/instance). The final-set evaluation of the key controls (Table 14) was added after the final set was opened and is accordingly *exploratory*: RL again significantly beats the random arm on both shells and the frequency-matched arm on TWBF-18 (on BBF-288 the frequency interval $[-0.167, +0.002]$ crosses zero, so no advantage is detected there); the open-loop stage table catches up and marginally exceeds RL on TWBF-18 ($+0.080 [+0.001, +0.162]$)—the closed-over-open-loop advantage replicates on the development set but not on the final distribution, and we say so.

5.8 Where the learned policy wins: a mechanism look

The 22 instances where the learned member improves on TWBF-18 span all three benchmark families (12 NT, 5 BKW, 5 C), with gains from 0.2 to 5.0 points; eleven of them (BKW02, C1P2, C7P2, N1A, N5D, N5E, N6B, N7C, T5D, T6D, T7B) also

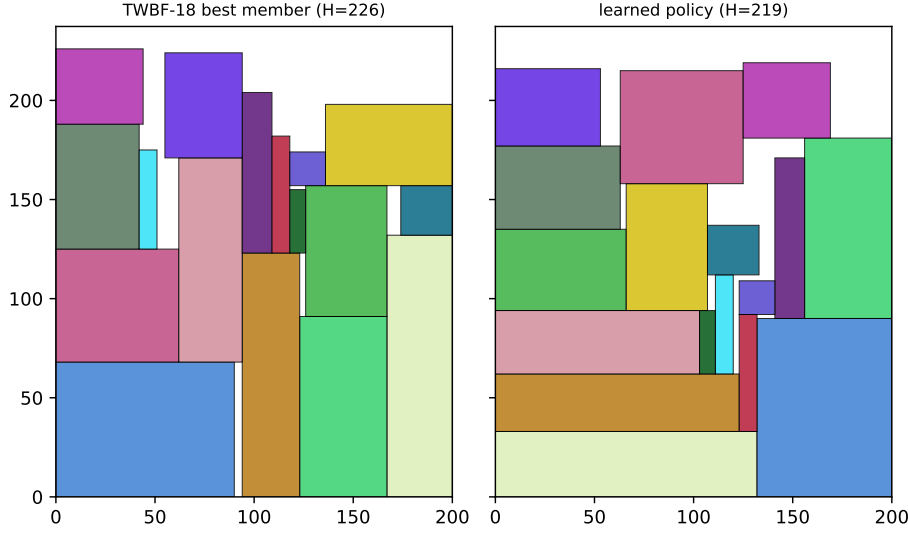


Fig. 4 Layouts on N1A: the TWBF-18 shell’s best member (left, $H=226$) vs the learned policy (right, $H=219$).

improve on BBF-288, a stable cross-shell subset. These are not small-margin ties. Figure 4 compares the final layouts on N1A (in the cross-shell subset); the layouts differ structurally, but final layouts alone cannot evidence mid-pass behaviour, so we restrict the claim to what the figure shows.

6 Limitations

(i) The raw learned constructor does not beat the best fixed configuration; the fourth-place standalone figure belongs to a two-member fallback ensemble. (ii) Composition gains are sub-half-point: statistically significant, practically modest. (iii) The learned member is never cost-competitive with budgeted local search at any budget tested; our cost claims are scoped to the compared implementations, data, and budgets. (iv) The observation is a compressed summary (POMDP); the reward telescopes to the pre-TR height, TR being an orthogonal shared postprocess. (v) An early pipeline version admitted a ± 0.3 -point scheduler-dependent jitter; all reported numbers regenerate from the fixed-order pipeline, and the artefact is disclosed in the package. (vi) Family holdout doubles as the development set for the ablation choices; the untouched final set (Section 5.7) bounds this. (vii) Evaluation is construction-and-budgeted-search only; long-run metaheuristics are a different cost class, not compared.

7 Conclusion

We asked what a learned controller adds when it composes classic strip-packing components step by step, sharing the components, controlling the comparisons, and counting

real compute cost. The evidence supports three layered conclusions. (1) *Complementary value*: the learned policy adds small but significant value to both published enumeration shells, replicating across training seeds and on the untouched final set. (2) *Mechanism, development set*: controls are consistent with state-dependent closed-loop decisions driving the gain—the policy significantly outperforms random switching, a stateless frequency-matched policy, and an open-loop stage table on both shells. (3) *Mechanism, final set*: this mechanism advantage did not replicate there—the open-loop stage table caught up (marginally ahead on TWBF-18, indistinguishable on BBF-288)—so we do not claim a general closed-loop advantage, only that the composition gain itself is robust. Existence of a composition gain and the role of state feedback are different claims; our evidence supports the first everywhere, and is consistent with a role for state-dependent decisions on the development set (our controls cannot establish necessity).

Equally central are the bounding facts: the raw policy trails the best train-selected fixed configuration; the augmented BBF-288 shell (4.405%) is numerically below FH but not significantly so, while a train-selected static 12-configuration portfolio from the same component space reaches 4.238% at one tenth of the augmented shell’s compute; and budgeted random-swap local search dominates bbF-288, FH, and both learning-augmented variants on both axes at every budget tested. Where learned stepwise composition adds value is now mapped precisely; so is where it does not. Replication artefacts (instances, episodes, per-instance results, trained models, analysis scripts, dated design log) accompany the paper.

8 Declarations

Funding. This work was supported by the Natural Science Foundation of Fujian Province (2024J011173).

Competing interests. The authors declare no competing interests.

Data availability. The replication package—code, instances, per-instance result CSVs, trained models, and analysis scripts—is public at <https://github.com/linwenru/stepwise-composition-strip-packing> and archived at <https://doi.org/10.5281/zenodo.23252641> (concept DOI 10.5281/zenodo.23252640). Key files (file → paper element): `rl-instance-gaps` (Table 2); `off3-combo` (Tables 3–4); `off3-controls` (Tables 5–6, Fig. 2 dev points); `off3-budget` (Table 7); `off3-ls-curve` (Table 8); `off4-timing*` (Table 9); `rl-cv*` (Table 10); `off3-multiseed` (Tables 11–12); `off3-final` (Table 13); `off3-round2` (Table 14, Fig. 2 final points); `off3-profile` (the thread-count artefact); `rl-models-*` (the trained per-fold models).

Acknowledgements

Work supported by the Natural Science Foundation of Fujian Province (2024J011173).

Author contributions. Wenru Lin: conceptualization, methodology, software, investigation, writing—original draft. Zhouwang Yang: supervision, writing—review & editing. Lei Liu: resources, validation.

Use of AI-assisted tools. An AI coding assistant was used to support code implementation and language editing of the manuscript. The research design, all experimental data, and the conclusions are solely the authors', who take full responsibility for the content and integrity of this work.

References

- [1] Aşık Ö, Özcan E (2009) Bidirectional best-fit heuristic for orthogonal rectangular strip packing. *Annals of Operations Research* 172:405–427. <https://doi.org/10.1007/s10479-009-0642-0>
- [2] Burke EK, Kendall G, Whitwell G (2004) A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research* 52(4):655–671. <https://doi.org/10.1287/opre.1040.0109>
- [3] Drake JH, Kheiri A, Özcan E, Burke EK (2020) Recent advances in selection hyperheuristics. *European Journal of Operational Research* 285(2):405–428. <https://doi.org/10.1016/j.ejor.2019.07.073>
- [4] Ernst D, Geurts P, Wehenkel L (2005) Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research* 6:503–556.
- [5] Fang J, Rao Y, Shi M (2023) A deep reinforcement learning algorithm for the rectangular strip packing problem. *PLoS ONE* 18(3):e0282598. <https://doi.org/10.1371/journal.pone.0282598>
- [6] Gabel T, Riedmiller M (2007) Reinforcement learning for scheduling: optimistic versus non-optimistic learning. *CISched/ITIC workshop papers*.
- [7] Garrido P, Riff MC (2007) An evolutionary hyperheuristic to solve strip packing problems. *IDEAL, LNCS* 4881:406–415. https://doi.org/10.1007/978-3-540-77226-2_42
- [8] Hopper E (2000) Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods. PhD thesis, Cardiff University.
- [9] Hopper E, Turton BCH (2001) An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research* 128(1):34–57.
- [10] Kaleta M, Śliwiński T (2026) Improving neural-based heuristics for 2D strip packing through local search in heuristic space. *Neural Computing and Applications*. <https://doi.org/10.1007/s00521-026-12013-2>
- [11] Kallestad J, Hasibi R, Hemmati A, Sørensen K (2023) A general deep reinforcement learning hyperheuristic framework for solving combinatorial optimization problems. *European Journal of Operational Research* 309:446–468. <https://doi.org/10.1016/j.ejor.2023.01.017>

- [12] Laterre A, Fu Y, Jabri MK, et al. (2018) Ranked reward: enabling self-play reinforcement learning for combinatorial optimisation. arXiv:1807.01672.
- [13] Li D, Ren Y, Gu J, Wang W, Lau H (2022) One model packs thousands of items with recurrent conditional query learning. Knowledge-Based Systems 235:107683. arXiv:2111.06726.
- [14] Li Y, Archetti C, Ljubić I (2024) Reinforcement learning approaches for the orienteering problem. Transportation Science. <https://doi.org/10.1287/trsc.2022.0366>
- [15] Li L, Liu B, Wu Z (2025) A backtracking heuristic algorithm for two-dimensional strip packing with rotation. Science Progress 108(1). <https://doi.org/10.1177/00368504241301530>
- [16] Li K, Lü Z, Ding J, Su Z, Zhang Q (2026) A skyline-guided bounded tree search framework with anticipative pruning for the 2-D strip packing problem with rotations. IEEE Transactions on Automation Science and Engineering 23:12174–12186. <https://doi.org/10.1109/TASE.2026.3708741>
- [17] López-Camacho E, Terashima-Marín H, Ross P, Ochoa G (2014) A unified hyper-heuristic framework for solving bin packing problems. Expert Systems with Applications 41(15):6876–6889. <https://doi.org/10.1016/j.eswa.2014.04.043>
- [18] Neuenfeldt Júnior A, Siluk JCM, Francescatto GP, Stieler E, Disconzi MA (2023) Algorithm selection for the two-dimensional strip packing problem with rotation. Expert Systems with Applications 213:119202. <https://doi.org/10.1016/j.eswa.2022.119202>
- [19] Ortmann FG, Ntene N, van Vuuren JH (2010) New and improved level heuristics for the rectangular strip packing and variable-sized bin packing problems. European Journal of Operational Research 203(2):306–315.
- [20] Piechowiak K, Drozdowski M, Sanlaville E (2022) Framework of algorithm portfolios for strip packing problem. Computers & Industrial Engineering 172:108538. <https://doi.org/10.1016/j.cie.2022.108538>
- [21] Rakotonirainy RG (2020) Machine learning for automated algorithm selection in strip packing. ORiON 36(2):73–88. <https://doi.org/10.5784/36-2-686>
- [22] Rakotonirainy RG, van Vuuren JH (2021) The effect of benchmark data characteristics during empirical strip packing heuristic performance evaluation. OR Spectrum 43:467–495.
- [23] Riedmiller M (2005) Neural fitted Q iteration—first experiences with a data efficient neural reinforcement learning method. ECML, LNCS 3720:317–328. https://doi.org/10.1007/11564096_32
- [24] Terashima-Marín H, Ross P, Farías-Zárate CJ, López-Camacho E, Valenzuela-Rendón M (2010) Generalized hyper-heuristics for solving 2D regular and irregular packing problems. Annals of Operations Research 179:369–392. <https://doi.org/10.1007/s10479-008-0475-2>

- [25] Tu Y, Bai R, Aickelin U, Zhang Y, Du W (2023) A deep reinforcement learning hyper-heuristic with feature fusion for online packing. *Expert Systems with Applications* 230:120568. <https://doi.org/10.1016/j.eswa.2023.120568>
- [26] Verstichel J, De Causmaecker P, Vanden Berghe G (2013) An improved best-fit heuristic for the orthogonal strip packing problem. *International Transactions in Operational Research* 20(5):711–730.
- [27] Wang Y, Peng Q, Wang Z, Huang S, Xu Z, Huang C, Guo B (2026) A hyper-heuristic algorithm based on Q-learning for open-dimension irregular packing. *Computers & Operations Research*. <https://doi.org/10.1016/j.cor.2025.107279>
- [28] Wei L, Oon WC, Zhu W, Lim A (2011) A skyline heuristic for the 2D rectangular packing and strip packing problems. *European Journal of Operational Research* 215(2):337–346.
- [29] Wei L, Hu Q, Leung S, Zhang N (2017) An improved skyline based heuristic for the 2D strip packing problem and its efficient implementation. *Computers & Operations Research* 80:113–127.
- [30] Xu Y, Yang Z (2024) GraphPack: a reinforcement learning algorithm for strip packing problem using graph neural network. *Journal of Circuits, Systems and Computers* 33(8):2450139. <https://doi.org/10.1142/S0218126624501391>
- [31] Zhang Y, Bai R, Qu R, Tu Y, Jin J (2022) A deep reinforcement learning based hyper-heuristic for combinatorial optimisation with uncertainties. *European Journal of Operational Research* 300:418–427. <https://doi.org/10.1016/j.ejor.2021.10.032>
- [32] Zhu K, Ji Z, Li Y (2020) A hybrid heuristic with reinforcement learning for strip packing. *IEEE Access* 8:226784–226796. <https://doi.org/10.1109/ACCESS.2020.3045905>
- [33] Leung SCH, Zhang D (2011) A fast layer-based heuristic for non-guillotine strip packing. *Expert Systems with Applications* 38(10):13032–13042.