Online supplement to:

Kirschenman, W., Heese, S., Kay, M., King, R., & McConnell, B. 2026. The 2-D Orthogonal Packing Problem with Multiple Levels of Prioritization: A Spatial Optimization Perspective, Forthcoming at *Naval Research Logistics*. Article in press.

Supplemental Materials: Adapted Methods for the Heuristic Benchmarking Study

This supplemental document describes the 14 adapted methods used in the heuristic benchmarking study. Its purpose is to make the benchmark methods easy to inspect and reproduce. We do not claim that these are the only relevant heuristics in the literature. Rather, we selected a broad set of common constructive, improvement, multi-start, and population-based procedures that could be adapted to the prioritization objective of 2DOPP-P and implemented under a common equal-budget design.

All methods use the same bin dimensions, feasibility rules, and objective function as the main manuscript. In these adapted implementations, the default access point is the center-left access point of the bin. The objective value of any complete layout is the main objective in Equation (1) of the main manuscript: the weighted Manhattan distance from each item centroid to the access point plus the weighted Manhattan distances between item centroids, with weights taken from the prioritization matrix π.

Several implementations use event-point candidate positions. In this context, an event point is generated from the origin or from the right and top edges of items already placed in the current layout. These candidate positions are then filtered for containment and non-overlap before objective-based scoring is applied.

In the pseudocode below, standard steps from the underlying method logic are shown in black. Problem-specific adaptations for 2DOPP-P are shown in blue.

Table S1 maps the benchmark labels used in the manuscript to the method descriptions provided below. The nine construction-plus-improvement methods are compositions: a constructor first creates a complete layout, and one of the three placement-space improvement routines is then applied to that layout.

Table S1: Mapping from benchmark labels to the method descriptions in these supplemental materials.

Benchmark label(s)	**Core procedure**	**Problem-specific adaptation**
BL-PS + LS BL-PS + VNS BL-PS + Tabu	Bottom-left constructor plus placement-space improver	Global priority is taken from the diagonal entries of π; final evaluation uses the main objective and center-left access point
MaxRects + LS MaxRects + VNS MaxRects + Tabu	MaxRects constructor plus placement-space improver	Items are processed in priority order; constructor scoring uses the diagonal access-point term; the main objective is used after construction and during improvement
SP+SA + LS SP+SA + VNS SP+SA + Tabu	Sequence-pair simulated annealing constructor plus placement-space improver	Sequence-pair states are evaluated by the main objective; rotations are part of the state
GRASP	Multi-start randomized MaxRects-style construction plus Local Search	Restricted candidate lists are built from incremental cost under the main objective using a fixed RCL threshold parameter
Reactive GRASP	Multi-start randomized MaxRects-style construction plus Local Search	Uses the same construction logic as GRASP, but updates the sampling probabilities of candidate RCL threshold values
GA + MaxRects	Generational permutation GA with MaxRects-style decoder	Chromosomes encode placement order; decoding uses incremental cost under the main objective against already placed items
Iterated Greedy	MaxRects-initialized destroy-repair search with Local Search	Removal is biased toward lower-priority items; reconstruction inserts removed items in descending priority order using incremental cost under the main objective
Filtered Beam Search	Beam search over partial layouts	Partial states are scored by the current value of the main objective; the implementation falls back to MaxRects if the beam search stalls before producing a complete layout

1 Bottom-Left with Priority Sorting (BL-PS) Constructor

Description. BL-PS is the simplest constructor in the comparison set. It first determines a single global item order from the manuscript's priority structure and then places items one at a time according to the classical bottom-left rule. This can be viewed as a priority-ordered packing baseline: the item sequence reflects the manuscript's prioritization logic, while the placement

rule itself remains a standard geometric heuristic.

Adaptation to 2DOPP-P. The problem-specific adaptation is the item order, not the geometry rule. Global priority is extracted from the diagonal entries of the prioritization matrix π, so higher-priority items are packed earlier. The implementation checks both allowed orientations and then selects the feasible position with smallest y-coordinate and, among ties, smallest x-coordinate. The constructor itself does not optimize the pairwise terms of the main objective during placement; the main objective is used to evaluate the completed layout and any downstream improvement phase.

Algorithm S1 BL-PS constructor used before LS, VNS, or Tabu Search

```
Input: items, bin dimensions, prioritization matrix π, rotation flag
Initialize the current layout to empty
Sort items by decreasing diagonal priority weight π_ii
for each item in that order do
    Enumerate the allowed orientations
    Generate event-point candidate positions from the origin and the current item edges
    Keep only positions that satisfy containment and non-overlap
    Select the feasible position with smallest y-coordinate and, among ties, smallest x-coordinate
    Place the item at that position
end for
Evaluate the completed layout with the main objective and the center-left access point
Return the layout and its objective value
```

2 Maximum Rectangles (MaxRects) Constructor

Description. Maximum Rectangles (MaxRects) (Jylänki, 2010) maintains a list of currently empty rectangles and places one item at a time into one of those free regions. In plain terms, it reasons with the remaining rectangular cavities in the bin rather than scanning the entire geometry from scratch after every placement. This makes it a more informed constructive rule than BL-PS while still remaining easy to explain.

Adaptation to 2DOPP-P. The implementation processes items in the same global priority order used by BL-PS. For each feasible orientation and free rectangle, it anchors the item at the lower-left corner of that free rectangle, computes the centroid implied by that anchored placement, and scores the candidate by the diagonal access-point term π_{ii} times the Manhattan distance from that centroid to the center-left access point. The center-left access point therefore enters through centroid-based candidate scoring, not through a different placement anchor. The pairwise group-cohesion terms enter the evaluation of the main objective and any subsequent improvement phase rather than the constructor score itself.

Algorithm S2 MaxRects constructor used before LS, VNS, or Tabu Search

```
Input: items, bin dimensions, prioritization matrix π, access point, rotation flag
Initialize the free-rectangle list with the full bin
Initialize the current layout to empty
Sort items by decreasing diagonal priority weight π_ii
for each item in that order do
    Initialize the best candidate score to +∞
    for each free rectangle do
        for each allowed orientation do
            if the item fits in the free rectangle then
                Place the candidate at the lower-left corner of the free rectangle
                Score the candidate by π_ii times its Manhattan distance to the center-left access point
                if the score improves the current best then
                    Store the candidate
                end if
            end if
        end for
    end for
    Insert the best candidate into the layout
    Split overlapping free rectangles around the placed item
    Remove redundant free rectangles
end for
Evaluate the completed layout with the main objective, including pairwise terms
Return the layout and its objective value
```

3 Sequence Pair with Simulated Annealing (SP+SA)

Description. A sequence pair (Murata et al., 1996) represents a layout indirectly through two item permutations. The pair determines which items must lie to the left of others and which must lie below others. Simulated annealing then explores this representation by making small random edits and occasionally accepting non-improving moves early in the search. This is a standard way to search fixed-outline floorplanning layouts without committing too early to explicit coordinates.

Adaptation to 2DOPP-P. The implementation evaluates each feasible decoded layout by the main objective, not by area or strip-height criteria. It also includes item rotations in the state and uses several deterministic initialization rules before falling back to random initialization attempts. In the benchmark table, this SP+SA layout serves either as a stand-alone constructor or as the starting point for LS, VNS, or Tabu Search.

Algorithm S3 Sequence Pair with Simulated Annealing (SP+SA)

Input: items, bin dimensions, prioritization matrix π, access point, iteration limit, cooling schedule
Build several initial sequence-pair states from priority order, width order, area order, and reversed-order variants
For each feasible decoded state, evaluate the main objective with the center-left access point
Keep the best feasible initial state
while the iteration limit is not reached **do**
    Generate a neighboring state by swapping adjacent elements in one sequence or toggling one rotation
    Decode the state into item coordinates
    if the decoded layout is feasible **then**
        Evaluate the main objective
        Accept the move if it improves the main objective or passes the simulated-annealing acceptance test
        Update the incumbent best state if needed
    end if
    Reduce the temperature according to the cooling schedule
end while
Return the best decoded layout and its objective value

4 Local Search

Description. Local Search improves an existing complete layout by checking small placement modifications. In the implementation used for this study, these modifications act directly on coordinates rather than on a separate encoding. That makes the method easy to pair with any constructor that already returns a feasible layout.

Adaptation to 2DOPP-P. The implementation uses three neighborhoods: anchor-point swaps, single-item reinsertion at event-point positions, and in-place 90-degree rotations when feasible. It applies a first-improvement strategy rather than an exhaustive best-improvement scan. Every accepted move is evaluated by the main objective.

Algorithm S4 Local Search improvement routine

```
Input: feasible layout, bin dimensions, prioritization matrix π, access point, iteration and time limits
Set the current layout to the input layout
repeat
    Search the swap neighborhood until the first improving feasible move is found
    if an improving swap is found then
        Accept it using the main objective
        Continue to the next iteration
    end if
    Search the insertion neighborhood by removing one item and testing event-point reinsertion positions
    if an improving insertion is found then
        Accept it using the main objective
        Continue to the next iteration
    end if
    if rotations are allowed then
        Search the in-place rotation neighborhood
        if an improving rotation is found then
            Accept it using the main objective
            Continue to the next iteration
        end if
    end if
until no improving move exists or a stopping limit is reached
Return the locally improved layout and its objective value
```

5 Variable Neighborhood Search

Description. Variable Neighborhood Search (VNS) (Mladenović and Hansen, 1997) escapes local optima by alternating between random perturbation and renewed descent. The idea is simple: if one neighborhood can no longer improve the current layout, deliberately move to a nearby but different layout and restart a local descent.

Adaptation to 2DOPP-P. The code used in this study employs three shaking neighborhoods that match the Local Search move types: swap, insertion, and rotation. After each shake, it does not limit the descent to the shaken neighborhood. Instead, it applies the full Local Search routine again. That shake-then-full-descent structure is the key implementation detail to document here.

Algorithm S5 Variable Neighborhood Search (VNS) improvement routine

Input: feasible layout, bin dimensions, prioritization matrix π, access point, iteration and time limits
Set the current and best layouts to the input layout
Define the shaking neighborhoods as random swap, random insertion, and random rotation
while the stopping limits are not reached **do**
    Set $k \leftarrow 1$
    while k does not exceed the number of neighborhoods **do**
        Apply one random shake in neighborhood k
        Run the full Local Search routine of Algorithm S4 on the shaken layout
        if the resulting layout improves the current one **then**
            Accept it, update the best layout if needed, and reset $k \leftarrow 1$
        else
            Set $k \leftarrow k + 1$
        end if
    end while
end while
Return the best layout found and its objective value

6 Tabu Search

Description. Tabu Search (Glover, 1989) evaluates a larger set of neighboring moves than simple Local Search and uses short-term memory to discourage cycling. The method is well suited to layout improvement because it can temporarily accept non-improving moves if that is needed to leave a local basin.

Adaptation to 2DOPP-P. The implementation evaluates feasible swap, insertion, and rotation moves in placement space and scores them by the main objective. A move signature is kept tabu for a finite number of iterations, while an aspiration rule allows tabu moves that improve the global best solution. The default tabu tenure is scaled to instance size.

Algorithm S6 Tabu Search improvement routine

Input: feasible layout, bin dimensions, prioritization matrix π, access point, iteration and time limits
Set the current and best layouts to the input layout
Initialize the tabu tenure, with default value $\max(7, \lfloor n/3 \rfloor)$
Initialize an empty tabu list
while the stopping limits are not reached **do**
    Enumerate feasible swap, insertion, and rotation moves
    Discard tabu moves unless the aspiration rule applies
    Score each admissible move by the main objective
    Apply the best admissible move
    Mark the move signature tabu for the chosen tenure
    Update the incumbent best layout if the new layout improves it
    Remove expired tabu entries periodically
end while
Return the best layout found and its objective value

7 Greedy Randomized Adaptive Search Procedure (GRASP) and Reactive GRASP

Description. Greedy Randomized Adaptive Search Procedure (GRASP) (Alvarez-Valdés et al., 2008) is a multi-start method with two phases in each iteration: randomized greedy construction and local improvement. In this implementation, the construction phase follows MaxRects-style free-rectangle logic rather than a simple placement scan. Instead of always taking the single cheapest construction move, it forms a restricted candidate list (RCL) of good moves and chooses randomly within that list. This creates diversification across repeated starts while preserving a consistent greedy bias.

Adaptation to 2DOPP-P. The construction phase is built on a MaxRects-style partial layout. At each step, the implementation scores all feasible item-placement-orientation candidates by their incremental cost under the main objective relative to the current partial layout, then forms an RCL using a threshold parameter α_{RCL}. Smaller values of α_{RCL} keep the RCL closer to the current best candidate, while larger values allow more diversification.

Standard GRASP keeps α_{RCL} fixed throughout the run. Reactive GRASP uses the same construction phase and the same Local Search phase, but samples α_{RCL} from a small candidate set and periodically updates those sampling probabilities from observed solution quality. This adaptive update is the key difference between the two variants.

Algorithm S7 GRASP and Reactive GRASP

Input: items, bin dimensions, prioritization matrix π, access point, time limit, Local Search limits
Initialize the incumbent best layout to null
For the reactive variant, initialize candidate values of α_{RCL} and their sampling probabilities
while the time or iteration limit is not reached **do**
    Choose α_{RCL} (fixed for standard GRASP, sampled for reactive GRASP)
    Initialize an empty partial layout and the corresponding free-rectangle list
    while unplaced items remain and at least one feasible candidate exists **do**
        Enumerate all feasible item-placement-orientation candidates
        Score each candidate by its incremental cost under the main objective against the current partial layout
        Form the restricted candidate list using the standard α_{RCL}-threshold rule
        Select one candidate uniformly at random from the restricted candidate list
        Insert the chosen candidate and update the free-rectangle list
    end while
    Apply Local Search to the constructed layout
    Update the incumbent best layout if improved
    For the reactive variant, periodically update the α_{RCL}-sampling probabilities from observed solution quality
end while
Return the best layout found and its objective value

8 Genetic Algorithm with MaxRects Decoder

Description. The genetic algorithm (Bortfeldt and Gehring, 2001) searches over item permutations rather than over coordinates directly. A chromosome specifies a placement order, and a decoder converts that order into a feasible layout. This separation lets the evolutionary search act on a simple representation while feasibility is enforced by the decoder.

Adaptation to 2DOPP-P. The implementation used in this study is generational rather than steady-state. Its initial population includes priority-ordered and area-ordered permutations plus random permutations. The decoder is MaxRects-style: it places each item in chromosome order and scores feasible candidate placements by incremental cost under the main objective relative to already placed items. Tournament selection, order crossover, and two mutation operators are used within a time-limited loop.

Algorithm S8 Genetic Algorithm with MaxRects-style decoder

```
Input: items, bin dimensions, prioritization matrix π, access point, time limit
Initialize the population with a priority permutation, an area permutation, and random permutations
for each chromosome in the population do
    Decode it into a feasible layout
    During decoding, score each candidate placement by incremental cost under the main objective relative to already placed items
end for
Store the incumbent best decoded layout
while the time limit is not reached do
    Select parents by tournament selection
    Create offspring by order crossover
    Mutate offspring by swap and insert moves
    for each offspring chromosome do
        Decode it into a layout and evaluate it
        Update the incumbent best layout if improved
    end for
    Replace the population with the offspring generation
end while
Return the best decoded layout and its objective value
```

9 Iterated Greedy

Description. Iterated Greedy (Ruiz and Stützle, 2007) alternates between partial destruction of the current solution and a greedy reconstruction of the missing part. In this implementation, the search starts from a MaxRects layout and then revisits only part of that layout at each iteration. This is useful when a strong constructive rule is already available, because the destroy-repair step can revisit a subset of decisions without having to restart from scratch each time.

Adaptation to 2DOPP-P. The implementation starts from a MaxRects layout. It removes a fraction of currently placed items, reconstructs them in descending priority order with in-

cremental cost under the main objective, and then applies Local Search. The removal is not uniform: items with smaller diagonal priority weights are more likely to be removed and rebuilt. Acceptance uses a simulated-annealing-style probability based on the current iteration count.

Algorithm S9 Iterated Greedy with destroy-repair and Local Search

```
Input: items, bin dimensions, prioritization matrix π, access point, time limit
Build an initial layout with the MaxRects constructor
Set the current and best layouts to that initial layout
while the time limit is not reached do
    Choose the destroy size
    Sample the removed items with probabilities inversely proportional to the diagonal priority weights, so lower-priority items are more likely to be rebuilt
    Delete those items from the current layout
    Reinsert the removed items in descending priority order using incremental cost under the main objective
    Apply Local Search to the reconstructed layout
    if the new layout improves the incumbent best then
        Update the incumbent best layout
    end if
    Accept or reject the new layout using the temperature-based Iterated Greedy acceptance rule
end while
Return the best layout found and its objective value
```

10 Filtered Beam Search

Description. Filtered Beam Search (Chen et al., 2025) builds the layout one item at a time but keeps several promising partial layouts in memory at each stage instead of committing to only one. In effect, it is a structured tree search over partial packings. The beam width controls how many partial layouts survive each expansion step.

Adaptation to 2DOPP-P. Items are introduced in descending priority order. Each beam state is expanded by feasible event-point placements of the next item, and partial states are scored by the current value of the main objective over the items already placed. The implementation sets the default beam width from instance size and includes a MaxRects fallback if the beam search terminates with an incomplete layout, so the benchmark always returns a complete feasible adapted-method solution when the fallback succeeds.

Algorithm S10 Filtered Beam Search

Input: items, bin dimensions, prioritization matrix π, access point, time limit, beam width
Initialize the beam with the empty partial layout
Sort items by decreasing diagonal priority weight π_{ii}
for each item in that order **do**
    Expand every beam state by all feasible event-point placements and allowed orientations of the next item
    Score each child state by the current value of the main objective over the items already placed
    Keep the best child states up to the beam width
    if the time limit is reached or no child state exists **then**
        Break
    end if
end for
if the best beam state does not contain all items **then**
    Run the MaxRects constructor as a fallback and return that layout if it is feasible
end if
Return the best complete layout obtained

References

Alvarez-Valdés, R., Parreño, F., Tamarit, J.M., 2008. Reactive GRASP for the strip-packing problem. Computers & Operations Research 35, 1065–1083. doi:10.1016/j.cor.2006.07.004.

Bortfeldt, A., Gehring, H., 2001. A hybrid genetic algorithm for the container loading problem. European Journal of Operational Research 131, 143–161. doi:10.1016/S0377-2217(00)00055-2.

Chen, M., Peng, X., Tang, X., 2025. Filtered beam search algorithm for the two-dimensional rectangular packing problem. International Transactions in Operational Research 32, 3729–3755. doi:10.1111/itor.70010.

Glover, F., 1989. Tabu search—part I. ORSA Journal on Computing 1, 190–206. doi:10.1287/ijoc.1.3.190.

Jylänki, J., 2010. A thousand ways to pack the bin: A practical approach to two-dimensional rectangle bin packing. URL: https://github.com/juj/RectangleBinPack/blob/master/RectangleBinPack.pdf. technical report.

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Computers & Operations Research 24, 1097–1100. doi:10.1016/S0305-0548(97)00031-2.

Murata, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y., 1996. Vlsi module placement based on rectangle-packing by the sequence-pair. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 15, 1518–1524. doi:10.1109/43.552084.

Ruiz, R., Stützle, T., 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177, 2033–2049. doi:10.1016/j.ejor.2005.12.009.