A Sahni-style Difficulty Measure for PortfolioFit

Project portfolio selection as geometric exact cover

Technical note - stouras.com/lab/portfoliofit - June 2026

PortfolioFit (built on the Ubongo packing mechanic) asks the player to fill a printed outline exactly, choosing from a set of dollar-valued polyomino "projects", with no gaps and no overlaps. Computer scientists classify this as a variant of the Exact Cover Problem and a two-dimensional Bin-Packing / Knapsack problem: a fixed "capacity" (the outline) must be packed with indivisible items (the bricks). This note transfers the Sahni-k difficulty measure, originally defined for the 0-1 knapsack problem, to PortfolioFit, and records two complementary measures the live game uses to grade its puzzles.

1. The knapsack / PortfolioFit correspondence

Feature	0-1 Knapsack Problem	PortfolioFit
Objective	Maximise the value of items packed into a bag without exceeding a weight limit.	Fill a shaded board region exactly with geometric pieces - no overlap, no spill.
Capacity	A scalar weight budget (e.g. 15 kg).	The exact cell-outline printed on the card (an area budget and a shape).
Items / pieces	Each item has a weight and a value.	A set of polyomino (tromino / tetromino / pentomino) tiles, each of fixed area.
Indivisibility	An item is taken whole (1) or not at all (0); no fractions.	Pieces cannot be cut; each is placed whole, with rotation / reflection.

The essential difference is dimensionality: knapsack feasibility is governed by a single scalar (total weight <= capacity), whereas PortfolioFit feasibility is governed by 2-D geometry (placements must tile a specific shape). PortfolioFit is therefore most precisely an instance of geometric exact cover, solvable with Knuth's Algorithm X / Dancing Links.

2. The Sahni-k difficulty of a knapsack instance

Sahni's (1975) approximation scheme S_k for the 0-1 knapsack works as follows: enumerate every subset of at most k items, force that subset into the bag, then complete the packing greedily in non-increasing value-to-weight ratio order; return the best packing found over all such subsets. As k grows the guarantee improves (relative error at most 1/(k+1)), and at k = n the method is exact.

The Sahni-difficulty of an instance is the smallest k for which S_k returns the optimum.

Intuitively, it is the fewest decisions one must commit by hand before a myopic greedy rule finishes the job correctly. A value k = 0 means greedy is already optimal (easy); a large k means greedy is repeatedly led into traps and substantial search is unavoidable (hard). Two ingredients make the measure work: (i) a canonical greedy rule, and (ii) the notion of "minimum forced decisions" to reach the target. Both port to PortfolioFit.

3. A Sahni-style difficulty for PortfolioFit

3.1 Instance

An instance is a region R of N cells together with the set of available bricks P = {p_1, ..., p_m} (here m = 8 - all bricks are available every round). A solution (tiling) places a SUBSET of the bricks - choosing an orientation (rotations and reflections allowed) and a translation for each - so that the

chosen placements partition R exactly; bricks that are not needed are left unused. This is exact cover by a sub-collection, and the player's job is to discover which subset fits.

3.2 Canonical greedy H (the value-to-cell ratio order)

Each brick carries a dollar value; its value-per-cell ratio - dollars divided by the number of cells it occupies - is the exact analogue of the knapsack value-to-weight ratio. The greedy H considers the bricks in non-increasing order of this ratio and places each at its first feasible spot (a fixed top-to-bottom, left-to-right cell scan) if it fits, otherwise skips it; it stops once the outline is filled. Deterministic and backtrack-free, it either tiles R (success) or jams - mirroring knapsack greedy producing a feasible but sub-optimal pack.

3.3 Definition

Let H-completion(S) denote the result of running H after pre-placing a set S of piece-placements. Write k(R,P) for the difficulty. Then:

```
k(R,P)  =  min over tilings T  of  min { |S| : S subset of T, and
H-completion(S) yields a full tiling }.
```

In words: k is the fewest correctly-placed pieces that must be revealed before a naive greedy player can finish the board without ever getting stuck. In the live game this is literally the minimum number of times the Hint button must be pressed before greedy play coasts home.

3.4 Properties (parallel to Sahni-k)

- k = 0: the greedy rule tiles the board outright - trivial.
- k = m: every piece is a trap; the seed must contain the whole solution - pure search.
- k forms a hierarchy 0, 1, ..., m and is always finite, because pinning all m pieces of any known tiling completes trivially (so k <= m).
- Computing k is itself the Sahni-style subset enumeration; for puzzle sizes with m <= 5 it is negligible (at most 2^m <= 32 greedy runs per tiling).

3.5 Algorithm

```
function sahniDifficulty(R, P):
    if H-completion(EMPTY) tiles R:        return 0        # k = 0
    best <- m
    for each tiling T of (R, P):                           # via Algorithm X / DLX
        for k = 1 .. best-1:
            for each subset S of T with |S| = k:
                if H-completion(S) tiles R:  best <- k; break
        if best == 1:  return 1                            # cannot beat k=1 once k != 0
    return best
```

Short-circuiting (test k = 0 first; stop at the first tiling that needs a single hint) keeps the computation fast in practice; solution enumeration is capped for robustness.

4. Two complementary measures

k captures trap structure (the Sahni spirit). Two further signals capture scarcity and human search effort, and are solver-light:

Measure	What it captures	Cost
k (Sahni analogue)	Minimum forced hints before greedy succeeds.	O(2^m . N)
Solution count |T|	Scarcity: a unique tiling is hardest; many tilings are forgiving.	One DLX count.

Search nodes / backtracks	Proxy for the effort a backtracking solver (or human) expends.	One solve.

The three agree at the extremes and diverge in the middle - the PortfolioFit version of "where are the hard knapsack problems": difficulty peaks when the number of tilings is small but positive and the pieces interlock (high k), not merely when the board is large.

5. Use in the live game

All eight bricks are available every round; the player must find the combination that fills the outline. Each brick has a dollar value, chosen so the eight value-per-cell ratios are all distinct (giving the greedy a strict order). The KPIs shown to the player mirror the Tetris app: value placed, the best value achievable on the board (the maximum-value exact cover), cells filled, and value density ($/cell).

Brick	I3	L	S	T	L5	Y	P	N
Cells	3	4	4	4	5	5	5	5
Value	$5	$6	$5	$7	$9	$6	$8	$7
$/cell	1.67	1.50	1.25	1.75	1.80	1.20	1.60	1.40

Difficulty is the value-ratio Sahni k itself. Because the outline is generated from a real tiling, a solution always exists; the game rejection-samples outlines until k matches the tier:

Tier	Outline	Defined by	Meaning
Easy	14 cells	k <= 1	the ratio-greedy almost solves it unaided
Hard	18 cells	k >= 3	the ratio-greedy is badly trapped - real search needed

References

S. Sahni, "Approximate algorithms for the 0/1 knapsack problem," Journal of the ACM 22(1):115-124, 1975.

D. E. Knuth, "Dancing Links," in Millennial Perspectives in Computer Science, 2000.

D. Pisinger, "Where are the hard knapsack problems?" Computers & Operations Research 32(9):2271-2284, 2005.