Before the Search Begins: Generating Function-Informed Initialization for Constrained Binary Optimization at Scale

Muhammad Marzouk Baig
MS Candidate, Khoury College of Computer Sciences
Northeastern University, Portland, Maine
baig.muham@northeastern.edu

Abstract

Population-based metaheuristics for constrained binary optimization require an initial population that is feasible, diverse, and reflective of the problem's constraint structure. The 0-1 multidimensional knapsack problem (MKP) is a canonical instance: the literature-standard probabilistic initialization [6] samples every item from a single global Bernoulli(p) determined by the active-variable fraction of the Toyoda greedy heuristic. We propose an alternative grounded in the combinatorial generating function tradition—the same algebraic framework Hardy and Ramanujan [9] used to analyze integer partitions. For each item j, the sub-generating function on the order-1 schema fixing item j enumerates how much of the feasible space survives when j is included versus excluded; proportional allocation of these feasibility densities yields a per-item inclusion probability p_j that is purely constraint-driven, requires no objective function, no LP relaxation, and no prior search, and degenerates automatically to uniform initialization when constraints are non-binding. We develop a simple Monte Carlo estimator using a Bernoulli(0.5) proposal—sufficient at small problem sizes ($n \leq 100$)—and a scalable extension based on self-normalised importance sampling (SNIS) with adaptive proposal-parameter selection and a restart-on-collapse mechanism that addresses weight degeneracy at high dimensions. On the full Chu–Beasley OR-Library benchmark (270 instances; n up to 500, m up to 30), the method beats the Hill baseline on 266 of 270 instances, achieves 100% gen-0 feasibility win-rate at medium and loose constraints and 95.6% at tight constraints, and characterizes the regime where the SNIS ratio estimator remains stable despite low effective sample size (ESS ≈ 2 at $n = 500$). We deliberately scope the paper to gen-0 initialization quality and make no claim about downstream GA convergence or final-solution quality.

1 Introduction

1.1 Motivation

Determining the population initialization approach is the first decision made when using a genetic algorithm on a binary optimization problem, yet it is too often treated as a formality. The standard approach—setting each bit independently to 1 with probability 0.5—is convenient and assumption-free, but it ignores the structure of the problem being solved. For unconstrained or loosely constrained problems this is harmless and the resulting random population is largely feasible. For tightly constrained combinatorial problems at scale, it is catastrophic: across all 90 tight-constraint OR-Library instances at $n \geq 100$ tested in this paper, uniform Bernoulli(0.5) initialization produces zero feasible solutions.

The generating function connection. Generating functions are a classical tool in combinatorics for encoding subset enumeration in compact algebraic form. The idea goes back at least to Euler and reached remarkable depth in the work of Hardy and Ramanujan [9] on integer partitions: the generating function $\prod_{k=1}^{\infty}(1-x^k)^{-1}$ encodes, in the coefficient of x^n, the number of ways to write n as a sum of positive integers. The same algebraic principle applies to our setting. For items with integer weights $w_1, \ldots, w_n$, the product $\prod_j(1+x^{w_j})$ encodes, in the coefficient of x^k, the number of subsets with total weight exactly k. This maps directly onto the feasibility question for constrained binary optimization: by factoring out one item at a time and evaluating the remainder against each capacity threshold, we can ask—before any search begins—how much of the feasible space survives when item j is included versus excluded. The closest prior work [6, 7] proposed using a single global inclusion probability derived from a greedy heuristic, applied identically to all items. Our approach instead derives per-item probabilities from sub-generating functions, requires no objective function, and generalizes to any number of constraints. To our knowledge, the generating function connection has not previously been exploited

for this purpose.

The two-phase algorithmic structure. The framework decomposes naturally into two algorithmic phases. At small problem sizes ($n \leq 100$), a simple Monte Carlo estimator with a Bernoulli(0.5) proposal estimates the per-item feasibility densities ρ_j^1 and ρ_j^0 directly. At larger sizes the Bernoulli(0.5) proposal becomes catastrophically inefficient: at $n = 250$ with $\alpha = 0.25$ the fraction of i.i.d. uniform draws that are feasible drops below 10^{-5} and both densities estimate to numerical zero across all items. We address this with a scalable extension: a Bernoulli(q) proposal whose sparsity is selected per instance by a short pilot procedure, a self-normalised importance sampling (SNIS) estimator, and a restart-on-collapse mechanism that bounds the effect of weight degeneracy at the very largest scales. The two phases share the same underlying mathematical framework; the IS extension is what makes the method usable at $n = 500$.

1.2 Scope

This paper characterizes initialization quality at generation zero. We do not claim that GF-biased initialization improves downstream GA convergence, time-to-target, or final solution quality. Those claims require running GAs to convergence under controlled conditions and depend on choices—repair operator, fitness model, crossover and mutation rates, selection scheme—that interact with initialization in non-trivial ways. We treat downstream GA behavior as a separate empirical question reserved for follow-up work.

1.3 Contributions

1. A generating function correspondence relating order-1 schemata to sub-generating functions over feasibility-weighted binary strings, yielding per-item feasibility densities ρ_j^1 and ρ_j^0 for any binary linear-constraint optimization instance (Section 3).
2. A proportional allocation rule $p_j = \rho_j^1/(\rho_j^1 + \rho_j^0)$ that recovers uniform initialization when constraints are non-binding and biases away from infeasibility-inducing items in proportion to the feasibility cost they impose (Section 3).
3. A simple Monte Carlo estimator with Bernoulli(0.5) proposal, GPU-vectorised and accurate to within 0.03% at $M = 10^7$, sufficient for $n \leq 100$ (Section 4.2).
4. A scalable extension based on a self-normalised importance sampling estimator with a Bernoulli(q) proposal, an adaptive q-selection pilot, and a restart-on-collapse mechanism that handles weight degeneracy at high dimensions (Section 4.3).
5. A constraint-only alternative proposal class—sequential random feasible greedy—suggested in personal correspondence [10] and integrated as a placeholder for empirical comparison in follow-up work (Section 4.4).
6. Empirical evaluation on the full Chu–Beasley OR-Library benchmark (270 instances; n up to 500, m up to 30), showing the method beats the Hill (1999) baseline on 266 of 270 instances at generation zero (Section 6).

1.4 Paper Organization

Section 2 reviews the MKP, the Chu–Beasley GA, schema theory, and generating functions. Section 3 develops the mathematical framework. Section 4 presents the exact and Monte Carlo algorithms, the IS-with-adaptive-q extension, the greedy-probe placeholder, and a complexity analysis. Section 5 provides worked examples on small instances. Section 6 presents the OR-Library benchmark results. Section 7 discusses applicability, limitations, and the broader scope of the framework. Section 8 concludes.

2 Background

2.1 The Multidimensional Knapsack Problem

The MKP generalizes the classical 0-1 knapsack problem to m constraints. Given n items each with value v_j and weight w_{ji} for constraint i, and m capacity constraints W_i, the MKP is:

$$\begin{aligned} &\text{maximize} && \sum_{j=1}^{n} v_j x_j \\ &\text{subject to} && \sum_{j=1}^{n} w_{ji} x_j \leq W_i \quad \forall i = 1, \ldots, m \\ &x_j \in \{0,1\} && \forall j = 1, \ldots, n \end{aligned}$$

The MKP is NP-hard and is a canonical benchmark for metaheuristics. Instances are commonly parameterized using the tightness ratio $\alpha = W_i / \sum_j w_{ji}$, which controls the fraction of the total item weight that fits in each constraint. Low α is tight; high α is loose. Standard benchmark instances are available from the OR-Library [2].

2.2 The Problem with Random Generation at Scale

Consider the MKP at $\alpha = 0.25$: each capacity is set to 25% of the total item weight across that constraint. On such instances the fraction of all binary strings that are feasible is tiny. Uniform random initialization samples from the full $\{0,1\}^n$ space without any awareness of where feasible solutions lie. Empirically, across all 90 OR-Library instances at $n \in \{100, 250, 500\}$,

$m \in \{5, 10, 30\}$ with $\alpha = 0.25$, the fraction of uniform-sampled solutions that are feasible is exactly zero. This is a structural consequence of initializing without any consideration of constraint structure, not an artifact of any particular fitness model or implementation choice.

2.3 Chu and Beasley Genetic Algorithm

Chu and Beasley [1] proposed a steady-state GA for the MKP that remains a standard baseline. Key components:

- **Initialization:** Each bit x_j set to 1 with probability 0.5 independently.
- **Crossover:** Uniform crossover.
- **Mutation:** Bit-flip at rate $1/n$.
- **Selection:** Binary tournament.
- **Repair:** A deterministic operator that iteratively drops items (in ascending order of efficiency u_j) until feasibility is restored, then greedily adds items (in descending order of u_j) while feasibility is maintained, where:

$$u_j = \frac{v_j}{\sum_{i=1}^{m} w_{ji}/W_i}$$

The repair operator is effective on standard benchmarks because item efficiency is a reliable proxy for solution quality when values and weights are uncorrelated. It is undefined when no objective exists and potentially misleading when value-weight correlation is adversarial [5].

2.4 Hill's Global-Density Initialization

Hill [6] proposed the first problem-informed initialization method for MKP GAs. The approach runs the Toyoda greedy heuristic to find a feasible solution, then uses the ratio of active variables ($x_j = 1$) in that solution as a single global $\Pr(X = 1)$, applied identically to all items. Hill and Hiremath [7] subsequently demonstrated that this global-density method accelerates GA convergence on two-dimensional knapsack problems, particularly on tight instances where the default $\Pr(X = 1) = 0.5$ produces few feasible solutions. Our work replaces the single global probability with per-item probabilities derived from generating functions, eliminating the dependence on objective values and generalizing to any number of constraints.

2.5 Schema Theory

Schema theory, introduced by Holland [3], provides a framework for analyzing GA behavior at the level of partial solutions. A schema is a string of length n over $\{0, 1, *\}$, where $*$ is a wildcard matching either value. An order-k schema has k fixed positions. The Schema Theorem states that short, low-order, above-average schemata are sampled exponentially more frequently across generations. Initialization determines the initial representation frequency of all schemata; biasing toward high-feasibility order-1 schemata directly seeds the population with the building blocks from which feasible high-quality solutions will be composed.

2.6 Generating Functions in Combinatorics

A generating function is a formal power series whose coefficients encode combinatorial quantities. The tradition is ancient: Euler used generating functions to analyze integer partitions, and the field reached a landmark in the Hardy–Ramanujan asymptotic formula [9] for the partition function $p(n)$, derived by analyzing $\prod_{k=1}^{\infty}(1-x^k)^{-1}$. For a finite set of items with integer weights $w_1, \ldots, w_n$, the analogous product is:

$$f(x) = \prod_{j=1}^{n}(1 + x^{w_j})$$

The coefficient of x^k in $f(x)$ equals the number of subsets with total weight exactly k. This follows from the distributive law: each factor contributes either 1 (item excluded) or x^{w_j} (item included), so the product enumerates all 2^n subsets. Wilf [4] provides a comprehensive treatment of generating functions in combinatorics; the application to constraint feasibility and GA initialization developed here is, to our knowledge, new.

3 Mathematical Framework

3.1 Feasibility and the Full Generating Function

Consider the MKP with n items and m constraints. A solution $\mathbf{x} \in \{0, 1\}^n$ is feasible if:

$$\sum_{j=1}^{n} w_{ji} x_j \leq W_i \quad \forall i = 1, \ldots, m$$

For m constraints, the full generating function over all 2^n subsets is:

$$f(x_1, \ldots, x_m) = \prod_{j=1}^{n} \left(1 + x_1^{w_{j1}} x_2^{w_{j2}} \cdots x_m^{w_{jm}}\right)$$

A term $x_1^{k_1} \cdots x_m^{k_m}$ in the expanded product corresponds to a feasible subset if and only if $k_i \leq W_i$ for all i.

3.2 Sub-Generating Functions and Feasibility Densities

To analyze feasibility at the schema level, we define the *sub-generating function* for item j as the generating function over all $n - 1$ remaining items:

$$g_j(x_1, \ldots, x_m) = \prod_{k \neq j} (1 + x_1^{w_{k1}} \cdots x_m^{w_{km}})$$

This function enumerates all 2^{n-1} subsets of items other than j. The function g_j is the same regardless of whether item j is fixed to 0 or 1; the distinction lies solely in the capacity threshold applied when counting feasible terms.

When item j is **included** ($x_j = 1$), its weight consumes capacity, leaving $W_i - w_{ji}$ for the remaining items. The feasibility density is:

$$\rho_j^1 = \frac{\#\{\text{terms in } g_j : \text{exp. of } x_i \le W_i - w_{ji}\ \forall i\}}{2^{n-1}}$$

When item j is **excluded** ($x_j = 0$), full capacity is available:

$$\rho_j^0 = \frac{\#\{\text{terms in } g_j : \text{exp. of } x_i \le W_i\ \forall i\}}{2^{n-1}}$$

Intuitively, ρ_j^1 is the probability that a uniformly random solution *containing* item j is feasible; ρ_j^0 is the same for solutions *not* containing j.

3.3 Order-1 Schemas and the Schema-Level Interpretation

A schema is a string over $\{0, 1, *\}$ where $*$ is a wildcard. An order-1 schema has exactly one fixed position. For n items there are $2n$ order-1 schemas—two per item—each containing exactly 2^{n-1} strings. The sub-generating function g_j corresponds exactly to the order-1 schema for item j: it enumerates and characterizes the feasibility of all 2^{n-1} strings that agree with item j's fixed value. Restricting analysis to order-1 structures requires n sub-generating function computations (one per item) rather than the 3^n required for full schema enumeration.

3.4 Sampling Probability

The initialization probability for bit j is allocated proportionally to the feasibility signal of each choice. For two mutually exclusive options with scores a and b, the proportional allocation rule assigns probability $a/(a+b)$ to option a. Applied to the two feasibility densities:

$$p_j = \frac{\rho_j^1}{\rho_j^1 + \rho_j^0}$$

This rule satisfies two important properties. First, $p_j + (1 - p_j) = 1$ by construction. Second, the ratio of probabilities equals the ratio of feasibility densities: if excluding item j is twice as feasibility-preserving as including it, the algorithm is twice as likely to exclude it.

Boundary cases. When $\rho_j^1 = \rho_j^0$, we have $p_j = 0.5$, recovering standard uniform initialization. This occurs on loose instances (α near 1), where the presence or absence of any single item has negligible effect on feasibility. When $\rho_j^1 = 0$, item j is never compatible with any completion of the remaining items—$p_j = 0$ and the item is never selected. When $\rho_j^0 = 0$, all completions are infeasible regardless of whether j is included or not, which cannot occur on any instance for which the empty solution is feasible.

4 Algorithm

4.1 Exact Computation

The exact procedure computes ρ_j^1 and ρ_j^0 by explicitly enumerating all 2^{n-1} terms of each sub-generating function. The full generating function f itself is never expanded; only the n sub-generating functions g_j are required.

Algorithm 1 GF-Informed Initialization (Exact)

Require: n items, m constraints, weights w_{ji}, capacities W_i, population size P
Ensure: Initial population of P solutions
Phase 1 — Precomputation (done once)
for $j = 1$ to n **do**
  Expand $g_j(x_1, \ldots, x_m) = \prod_{k \ne j} \left(1 + x_1^{w_{k1}} \cdots x_m^{w_{km}}\right)$
  $\rho_j^1 \leftarrow$ fraction of terms where exp. of $x_i \le W_i - w_{ji}\ \forall i$
  $\rho_j^0 \leftarrow$ fraction of terms where exp. of $x_i \le W_i\ \forall i$
  $p_j \leftarrow \rho_j^1/(\rho_j^1 + \rho_j^0)$
end for
Phase 2 — Population Initialization
for $s = 1$ to P **do**
  for $j = 1$ to n **do**
    $x_j \leftarrow 1$ with probability p_j, else 0
  end for
  Add solution $\mathbf{x}$ to population
end for
return population

Exact computation requires enumerating 2^{n-1} terms per item and is feasible only for $n \le 20$ in practice.

4.2 Simple Monte Carlo with Bernoulli(0.5)

For practical instances ($n \approx 50$–100), exact enumeration is intractable. The simplest Monte Carlo estimator approximates ρ_j^1 and ρ_j^0 by drawing M random binary vectors $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(M)}$ over the $n-1$ items other than j, with each $s_\ell^{(k)} \overset{\text{i.i.d.}}{\sim}$ Bernoulli(0.5), and counting the fraction feasible under each threshold:

$$\hat{\rho}_j^1 = \frac{1}{M} \sum_{k=1}^{M} \mathbf{1}\left[\sum_{\ell \ne j} w_{\ell i}\, s_\ell^{(k)} \le W_i - w_{ji}\ \ \forall i\right]$$

$$\hat{\rho}_j^0 = \frac{1}{M} \sum_{k=1}^{M} \mathbf{1}\left[\sum_{\ell \ne j} w_{\ell i}\, s_\ell^{(k)} \le W_i\ \ \forall i\right]$$

Accuracy. Each indicator is a Bernoulli random variable, so the standard error of $\hat{\rho}_j^1$ is $\sqrt{\rho_j^1(1-\rho_j^1)/M} \le 1/(2\sqrt{M})$. At $M = 10^7$, this upper bound is 0.016%.

GPU acceleration. The $M \times (n-1)$ matrix of random binary samples is generated and evaluated entirely on a GPU using PyTorch tensor operations. On a T4 GPU, computing all n probability estimates for $n = 100$, $m = 3$, $M = 10^7$ takes 5–8 seconds.

The failure mode that motivates 4.3. At larger problem sizes with tight constraints ($n \geq 250$, $\alpha = 0.25$), the Bernoulli(0.5) proposal becomes catastrophically inefficient. A typical Bernoulli(0.5) draw selects $\sim n/2$ items with total weight far exceeding the capacity $W_i = \alpha \sum_j w_{ji} \approx 0.25 \sum_j w_{ji}$. The fraction of Bernoulli(0.5) samples that are feasible drops below 10^{-5}, and both $\hat{\rho}_j^1$ and $\hat{\rho}_j^0$ estimate to numerical zero across all items. The estimator returns nothing usable. This is not a flaw in the proportional allocation rule of Section 3.4; it is a property of the proposal distribution. The scaled extension below fixes the proposal.

4.3 Importance Sampling with Adaptive q

For instances where Bernoulli(0.5) collapses, we substitute a Bernoulli(q) proposal and use self-normalised importance sampling. The proposal sparsity q is selected per instance by a short pilot procedure, and a restart-on-collapse mechanism bounds the effect of weight degeneracy at high dimensions.

4.3.1 The SNIS estimator

We estimate p_j directly by drawing N samples $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(N)}$ from Bernoulli(q), computing IS weights

$$w^{(i)} = \prod_{j=1}^{n} \left[q^{s_j^{(i)}} (1-q)^{1-s_j^{(i)}} \right]^{-1}$$

and forming the SNIS estimator

$$\hat{p}_j = \frac{\sum_i w^{(i)} \cdot s_j^{(i)} \cdot f(\mathbf{s}^{(i)})}{\sum_i w^{(i)} \cdot f(\mathbf{s}^{(i)})}$$

where $f(\cdot)$ is the feasibility indicator.

Why this estimator is stable at low ESS. Self-normalisation is the operative reason this method scales. Numerator and denominator are sums over the *same* samples with the *same* weights $w^{(i)}$. Noise that inflates a small set of heavy-weight samples tends to inflate both the numerator and denominator in the same direction, and the ratio absorbs most of it. The variance of $\hat{p}_j$ is consequently much smaller than the variance of either $\sum_i w^{(i)} \cdot s_j^{(i)} f(\mathbf{s}^{(i)})$ or $\sum_i w^{(i)} f(\mathbf{s}^{(i)})$ on its own. An alternative estimator—computing $\hat{\rho}_j^1$ and $\hat{\rho}_j^0$ as *separate* IS quantities and forming their ratio post hoc—would not share weights between numerator and denominator and would have catastrophic variance under the weight-degeneracy conditions of Section 6.6. The decision to estimate p_j directly as a single SNIS ratio is therefore not a convenience but a prerequisite for the method to function at $n = 500, \alpha = 0.25$ where the effective sample size is order unity.

Implementation. Log-domain weight computation for numerical stability; GPU batching at 10^5 samples per batch on T4; numerator and denominator accumulated separately and divided after batches complete; effective sample size ESS $= (\sum w)^2 / \sum w^2$ tracked per batch for diagnostics.

4.3.2 Adaptive q via pilot

The optimal proposal sparsity q depends on instance-specific feasibility geometry. We select q^* empirically via a short pilot:

1. Build 4 candidate values $\{\alpha - 0.10,\ \alpha - 0.05,\ \alpha,\ \alpha + 0.05\}$, clipped to $[0.05, 0.95]$.
2. For each candidate, run a small IS estimation (5×10^5 samples).
3. Sample a tiny population (2×10^3) from the resulting $\hat{p}_j$ vector.
4. Measure gen-0 feasibility on the 2K population.
5. Select $q^* = \arg\max$(gen-0 feasibility).

The pilot is an end-to-end mini-test, not an indirect diagnostic: it does not select on ESS or proposal feasibility, but on the actual quantity of interest (gen-0 feasibility of the resulting population). Pilot overhead is approximately 20% of the main run cost.

4.3.3 Restart-on-collapse

At high dimensions ($n \geq 500$) with tight constraints ($\alpha = 0.25$), the IS weight distribution becomes degenerate: a small number of samples may dominate all weight, producing a $\hat{p}_j$ vector that collapses to extremes ($\hat{p}_j \approx 0$ or $\hat{p}_j \approx 1$) and yields near-deterministic populations. We observe empirically that this failure is bimodal across random seeds—most seeds produce a useful $\hat{p}_j$, while a minority produce degenerate estimates.

Restart-on-collapse: for each instance, attempt up to $K = 5$ independent runs (different RNG seeds). For each attempt:

- Run the adaptive pilot.
- If pilot gen-0 feasibility $< 1\%$ (collapse threshold), skip the main run and retry—this saves the cost of a doomed 10^7-sample run.
- Otherwise run the main 10^7-sample IS estimation.
- If main gen-0 feasibility $\geq 1\%$, return the result.
- Otherwise retry.

The mechanism is bounded: at most $5\times$ compute cost on the hardest instances, $0\times$ cost on easy instances. In our experiments, restart fired on only 13 of 270 instances,

and only in two cells (mknapcb3 and mknapcb6 at $\alpha = 0.25$).

4.3.4 Algorithm pseudocode

Algorithm 2 IS-Based GF Initialization with Adaptive q and Restart-on-Collapse

Require: weights W, capacities c, tightness α, retry cap $K = 5$, collapse threshold $\tau = 0.01$
Ensure: probability vector $\hat{p}_j$

for attempt $= 1$ to K **do**
  seed RNG with attempt index

  Pilot — choose q^* by direct gen-0 test
  candidate_qs $\leftarrow$ clip($[\alpha - 0.10,\ \alpha - 0.05,\ \alpha,\ \alpha + 0.05]$, 0.05, 0.95)
  best_q $\leftarrow$ none; best_feas $\leftarrow 0$
  for q in candidate_qs **do**
    pj_pilot $\leftarrow$ SNIS_estimate(W, c, q, $N = 5 \times 10^5$)
    pilot_feas $\leftarrow$ sample_and_check(pj_pilot, W, c, pop = 2,000)
    if pilot_feas $>$ best_feas **then**
      best_q $\leftarrow q$; best_feas $\leftarrow$ pilot_feas
    end if
  end for

  Collapse check — skip doomed main run
  if best_feas $< \tau$ **then**
    continue ▷ retry with fresh seed
  end if

  Main run — full budget at chosen q
  pj_main $\leftarrow$ SNIS_estimate(W, c, best_q, $N = 10^7$)
  main_feas $\leftarrow$ sample_and_check(pj_main, W, c, pop = 10,000)
  if main_feas $\geq \tau$ **then**
    return pj_main
  end if ▷ else: main collapsed, retry
end for
return α-uniform fallback ▷ rare

Algorithm 3 Random Feasible Greedy Probe (placeholder)

Require: weights W, capacities c, n items, trials T
Ensure: global density $\hat{q}$ and per-item $\hat{p}_j$

inclusion_count $\leftarrow$ zeros(n)
items_per_trial $\leftarrow$ empty list
for $t = 1$ to T **do**
  available $\leftarrow \{1, 2, \ldots, n\}$
  remaining $\leftarrow c$ ▷ remaining capacity vector
  selected $\leftarrow$ empty set
  while available is nonempty **do**
    fits $\leftarrow \{j \in$ available $: w_j \leq$ remaining elementwise$\}$
    if fits is empty **then**
      break
    end if
    pick j uniformly at random from fits ▷ random, not value-ranked
    add j to selected; remove j from available
    remaining $\leftarrow$ remaining $- w_j$
  end while
  append |selected| to items_per_trial
  for j in selected **do**
    inclusion_count[j] $\leftarrow$ inclusion_count[j] $+1$
  end for
end for

$\hat{q} \leftarrow$ mean(items_per_trial) $/\ n$ ▷ Hill-style global
$\hat{p}_j \leftarrow$ inclusion_count[j] $/\ T$ for $j = 1, \ldots, n$ ▷ per-item extension
return $\hat{q}$, $\hat{p}$

4.4 Alternative Proposal: Random Feasible Greedy Probe

The Bernoulli(0.5) proposal of Section 4.2 and the adaptive Bernoulli(q) proposal of Section 4.3 are both i.i.d. proposals over the full hypercube. An alternative proposal class—suggested by Hill in personal correspondence [10]—generates samples sequentially via a random feasible greedy construction. We include the algorithmic landscape here for completeness; the section is a placeholder for empirical comparison in follow-up work, since experimental validation against the SNIS estimator of Section 4.3 on the OR-Library benchmark is not yet complete.

Constraint-only justification. Hill's original 1999 initialization uses the *Toyoda* greedy heuristic, which ranks items by an efficiency ratio that depends on item values v_j. By contrast, Algorithm 3 selects candidates uniformly at random from the currently-feasible set at each step—it uses only the weight matrix W and the capacities c, never item values. The proposal is therefore in the same "no objective information" class as the Bernoulli(0.5) proposal of Section 4.2 and the Bernoulli(q) proposal of Section 4.3, and is consistent with the constraint-only premise of the paper.

Open questions. Whether the per-item inclusion frequencies of the probe match or exceed the SNIS $\hat{p}_j$ estimator on the OR-Library benchmark; whether the probe used as an IS *proposal* (rather than as a direct estimator) yields higher ESS than Bernoulli(q) at high dimensions; whether the sequential cost ($O(n^2 m)$ per trial worst case) can be batched GPU-efficiently—all open. Empirical comparison is deferred to follow-up work.

4.5 Complexity Analysis

Exact computation. Expanding each sub-generating function g_j enumerates 2^{n-1} terms, and checking feasibility against m constraints requires $O(m)$ work per term. Phase 1 costs $O(n \cdot 2^n \cdot m)$—intractable for $n > 20$.

Simple Monte Carlo (Bernoulli(0.5)). Each of n items requires M sample evaluations, each $O(m)$. Phase 1 costs $O(n \cdot M \cdot m)$, linear and fully GPU-parallelizable

across the M sample dimension. Fails by proposal collapse at $n \geq 250$ with $\alpha = 0.25$.

IS with adaptive q. Same asymptotic complexity $O(n \cdot M \cdot m)$ but with log-domain weight bookkeeping. Pilot overhead is approximately 20%. Per-instance wall time on T4: $\sim$3 s at $n = 100$; $\sim$10 s at $n = 250$; $\sim$30 s at $n = 500$ (single attempt).

Random feasible greedy probe. Sequential, $O(T \cdot n^2 \cdot m)$ worst case per trial. Not directly GPU-parallelizable across samples; can be batched across trials.

Phase 2 (population initialization) costs $O(P \cdot n)$ in all cases and is dominated by Phase 1 for any reasonable population size P.

5 Worked Examples

Both examples in this section are computed analytically by expanding the generating functions explicitly. They are reproducible without any code and are included to build intuition before the larger experimental section.

5.1 Single-Constraint Example

Consider the MKP with $n = 3$ items and $m = 1$ constraint:

	Item 1	Item 2	Item 3
Weight	3	2	4
Value	5	4	3

Table 1. *3-item single-constraint instance. Capacity $W = 5$.*

Step 1 — Full generating function.

$$f(x) = (1 + x^3)(1 + x^2)(1 + x^4)$$
$$= 1 + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^9$$

Of the $2^3 = 8$ subsets, the feasible ones (exponent ≤ 5) are: $\emptyset, \{2\}, \{1\}, \{3\}, \{1, 2\}$—overall feasibility density $5/8 = 0.625$.

Step 2 — Sub-generating functions.

Item 1 ($w_1 = 3, W - w_1 = 2$): $g_1(x) = (1+x^2)(1+x^4) = 1 + x^2 + x^4 + x^6$. 2 of 4 terms have exponent ≤ 2; 3 of 4 have exponent ≤ 5.

$$\rho_1^1 = 0.50 \quad \rho_1^0 = 0.75 \quad p_1 = 0.40$$

Item 2 ($w_2 = 2, W - w_2 = 3$): $g_2(x) = (1+x^3)(1+x^4) = 1 + x^3 + x^4 + x^7$.

$$\rho_2^1 = 0.50 \quad \rho_2^0 = 0.75 \quad p_2 = 0.40$$

Item 3 ($w_3 = 4, W - w_3 = 1$): $g_3(x) = (1+x^3)(1+x^2) = 1 + x^2 + x^3 + x^5$.

$$\rho_3^1 = 0.25 \quad \rho_3^0 = 1.00 \quad p_3 = 0.20$$

Step 3 — Summary.

	Item 1	Item 2	Item 3
Weight	3	2	4
ρ_j^1	0.50	0.50	0.25
ρ_j^0	0.75	0.75	1.00
p_j **(ours)**	**0.40**	**0.40**	**0.20**
p_j (uniform)	0.50	0.50	0.50

Table 2. *Per-item densities and sampling probabilities.*

Item 3 consumes 80% of the capacity upon inclusion; the framework pushes p_3 to 0.20. Enumerating all 8 strings explicitly with these p_j values, the expected feasibility rate under our method is **87.2**% versus 62.5% under uniform initialization—a gain of 24.7 percentage points.

5.2 Multi-Constraint Example

Consider $n = 4$ items, $m = 2$ constraints, $W_1 = 7$, $W_2 = 8$:

	Item 1	Item 2	Item 3	Item 4
Weight (C1)	2	4	3	5
Weight (C2)	3	2	5	1
Value	4	7	5	3

Table 3. *4-item two-constraint instance.*

Of the $2^4 = 16$ possible solutions, 9 are feasible (overall density 56.25%). We work through Item 4 to illustrate the multi-constraint threshold check.

Sub-generating function for Item 4 ($w_{41} = 5, w_{42} = 1$):

$$g_4(x_1, x_2) = (1 + x_1^2 x_2^3)(1 + x_1^4 x_2^2)(1 + x_1^3 x_2^5)$$

Included: remaining capacity $(W_1 - 5, W_2 - 1) = (2, 7)$. Excluded: full $(7, 8)$.

$\{1,2,3\}$ subset	e_1	e_2	Incl	Excl
$\emptyset$	0	0	✓	✓
$\{3\}$	3	5	✗	✓
$\{2\}$	4	2	✗	✓
$\{2,3\}$	7	7	✗	✓
$\{1\}$	2	3	✓	✓
$\{1,3\}$	5	8	✗	✓
$\{1,2\}$	6	5	✗	✓
$\{1,2,3\}$	9	10	✗	✗

Table 4. *All 8 completions of g_4. e_1, e_2 are the total weights of the subset of $\{1, 2, 3\}$. Incl: $e_1 \leq 2, e_2 \leq 7$. Excl: $e_1 \leq 7, e_2 \leq 8$.*

From Table 4: 2 of 8 completions feasible when Item 4 is included; 7 of 8 feasible when it is excluded.

$$\rho_4^1 = 0.250 \quad \rho_4^0 = 0.875 \quad p_4 = 0.222$$

Item 4 consumes 71% of W_1 on its own; the constraint signal comes specifically from C1. The two-dimensional threshold check identifies automatically which constraint binds.

The same procedure for all four items:

	Item 1	Item 2	Item 3	Item 4
Weight (C1)	2	4	3	5
Weight (C2)	3	2	5	1
ρ_j^1	0.500	0.375	0.375	0.250
ρ_j^0	0.625	0.750	0.750	0.875
p_j **(ours)**	**0.444**	**0.333**	**0.333**	**0.222**
p_j (uniform)	0.500	0.500	0.500	0.500

Table 5. *Per-item probabilities, multi-constraint example.*

Enumerating all 16 strings under these p_j values, the expected feasibility rate under our method is **83.8**% versus 56.25% under uniform—a gain of 27.6 percentage points. Note that Items 2 and 3 are penalized equally ($p_j = 0.333$) despite different weight profiles: Item 2 is heavy on C1 and light on C2, Item 3 the reverse. Both are constrained equally by the *combination* of the two thresholds.

6 Experiments and Results

6.1 Setup

Benchmark. OR-Library `mknapcb1` through `mknapcb9` [1, 2], 30 instances per file, 270 instances total.

File	n	m	Instances
mknapcb1	100	5	30
mknapcb2	250	5	30
mknapcb3	500	5	30
mknapcb4	100	10	30
mknapcb5	250	10	30
mknapcb6	500	10	30
mknapcb7	100	30	30
mknapcb8	250	30	30
mknapcb9	500	30	30

Table 6. *OR-Library benchmark structure. Each file contains 10 instances at each of* $\alpha \in \{0.25, 0.50, 0.75\}$.

Baselines. (i) Uniform Bernoulli(0.5) — constraint-blind baseline. (ii) Hill / Toyoda greedy [6] — literature-standard probabilistic initialization; global $\Pr(X = 1)$ from Toyoda active-variable fraction applied identically to all items. Construction heuristics (random feasible greedy, Sahni's algorithm, surrogate constraint methods) are excluded from the main comparison since they fuse initialization with a specific repair operator; see Section 7.4.

Metrics. Per (method, instance), on a sampled population of 10^4: gen-0 feasibility (primary), val_max, items selected, Hamming diversity, unique solutions. GF-specific: $\mathrm{ESS}_{\mathrm{mean}}, \mathrm{ESS}_{\mathrm{min}}, q^*$, restart attempts, population sensitivity at sizes $\{100, 500, 1000, 5000, 10000\}$.

We do not report convergence-speed or final-quality metrics from a downstream GA. Those metrics depend on repair operator, fitness model, crossover/mutation parameters, and selection scheme, and are deliberately out of scope; see Section 7.4.

Reproducibility. Single NVIDIA T4 GPU (Google Colab), PyTorch 2.x, deterministic seeds (attempt index used as seed), 122 min total wall time for the full 270-instance sweep.

6.2 Headline Results

GF-biased initialization beats the Hill baseline on **266 of 270 instances**: 100% win-rate at $\alpha \in \{0.50, 0.75\}$ and 95.6% at $\alpha = 0.25$. Uniform Bernoulli(0.5) produces exactly 0% feasibility on all 90 tight-constraint instances. At tight constraints GF produces roughly four times more feasible solutions than Hill while preserving val_max within 1.6% of the Hill baseline; the val_max degeneracy at $\alpha = 0.75$ is a structural property of constraint-only methods sampling at $p_j \approx 0.5$ when Hill's value-aware Toyoda heuristic packs higher-value items more densely.

6.3 Per-Cell Breakdown

Selected highlights (full table in supplementary material, source `paper_table.csv`):

Tight ($\alpha = 0.25$): mknapcb1 ($n = 100, m = 5$): UN 0.00% / Hill 7.48% / GF **49.86**% (6.7× over Hill). mknapcb3 ($n = 500, m = 5$): UN 0.00% / Hill 0.27% / GF **44.77**% (163×). mknapcb9 ($n = 500, m = 30$): UN 0.00% / Hill 11.38% / GF **31.82**% (2.8×).

Medium ($\alpha = 0.50$): GF wins 10/10 instances in all 9 cells. mknapcb3: UN 26.34% / Hill 2.52% / GF **69.47**%.

Loose ($\alpha = 0.75$): GF degenerates to uniform; all 9 cells achieve ∼100% feasibility (matching uniform). Hill underperforms (∼ 17% mean) because its Toyoda-derived high global density samples populations whose total weight exceeds capacity more often.

6.4 The 4 Losses

GF loses to Hill on 4 of 270 instances, all at $\alpha = 0.25$:

α	GF wins / 90	UN feas	Hill feas	GF feas	GF val_max / Hill
0.25	**86 / 90**	0.00%	9.94%	**38.98%**	1.016
0.50	**90 / 90**	19.79%	13.43%	**69.19%**	1.000
0.75	**90 / 90**	99.997%	17.35%	**99.997%**	0.813
Total	**266 / 270**				

Table 7. *Aggregate results by tightness regime. "GF val_max / Hill" is GF val_max / Hill val_max averaged over the cell. At $\alpha = 0.75$ GF degenerates to uniform sampling (Section 3.4), so its val_max coincides with the uniform val_max which is below the value-aware Hill val_max.*

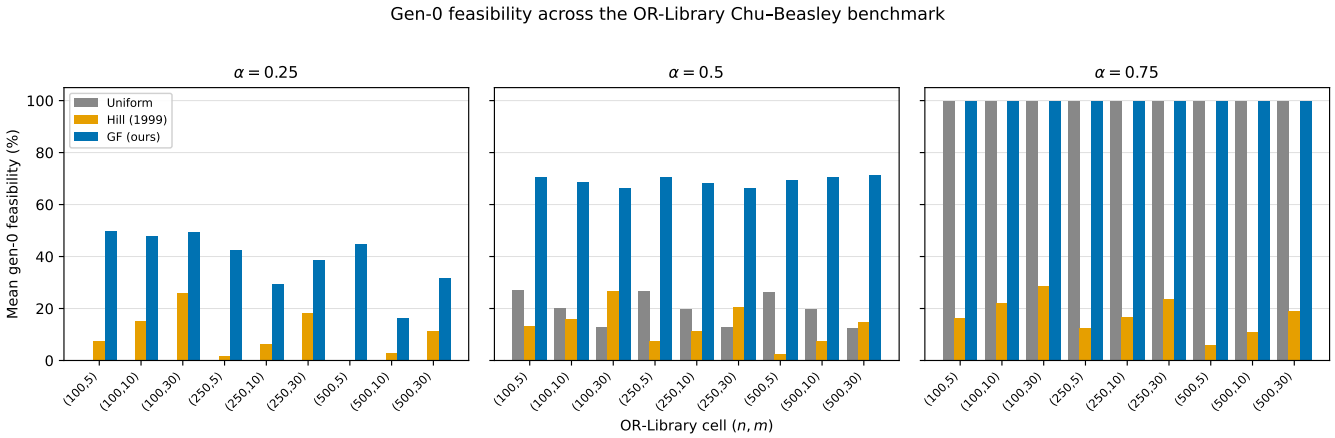

Figure 1. *Mean gen-0 feasibility across 10 instances per cell, 9 cells, 3 methods, faceted by α. Uniform Bernoulli(0.5) achieves 0% feasibility on all tight-constraint instances at $n \geq 100$. Hill [6] achieves 0.3–26% depending on cell. GF-biased achieves 16–50% at $\alpha = 0.25$ and 66–71% at $\alpha = 0.50$. At $\alpha = 0.75$ (loose constraints), GF correctly degenerates to uniform sampling.*

File	Inst	n	m	GF	Hill	$\text{ESS}_{\min}$
mknapcb5	5	250	10	3.4%	6.2%	1.0
mknapcb6	4	500	10	1.3%	3.9%	1.0
mknapcb8	9	250	30	12.8%	23.5%	1.0
mknapcb9	8	500	30	4.2%	13.6%	1.0

Table 8. *All four GF losses, with $\text{ESS}_{\min}$ diagnostic. All four share the worst-case weight degeneracy condition.*

All four losses share $\text{ESS}_{\min} = 1$ (worst-case weight degeneracy) and have gen-0 feasibility just above the 1% restart threshold, so restart did not trigger. A threshold of 5% would have recovered all four; we report 1% as a deliberately conservative choice and identify adaptive threshold selection as future work (Section 8.1).

6.5 Restart-on-Collapse Impact

Restart fired on 13 of 270 instances, concentrated in two cells. Detail for mknapcb3 ($n = 500, m = 5, \alpha = 0.25$), the cell where restart was decisive:

Configuration	Mean gen-0 feas	GF wins vs Hill
Without restart	4.04%	3 / 10
With restart-on-collapse	**44.77%**	**10 / 10**

Table 9. *Effect of restart-on-collapse on mknapcb3 at $\alpha = 0.25$. Restart turns a losing cell into a winning one.*

On 257 of 270 instances restart fires zero times; on the 13 where it fires, the pilot's early-stop saves approximately 420 seconds of compute on doomed main runs across the affected cells.

6.6 SNIS–ESS Decoupling

At $n = 500$ with $\alpha = 0.25$, the IS estimator's mean ESS is ≈ 2 and minimum ESS is ≈ 1 across all 30 instances per file. By standard IS diagnostics, the estimator is degenerate.

Yet GF gen-0 feasibility on these same instances ranges from 0% to 95.7%, with cell-mean values of 31–45%. The decoupling arises from the SNIS structure: numerator and denominator share the same weights, and noise inflating one tends to inflate the other in the same direction. The ratio has lower variance than either component alone.

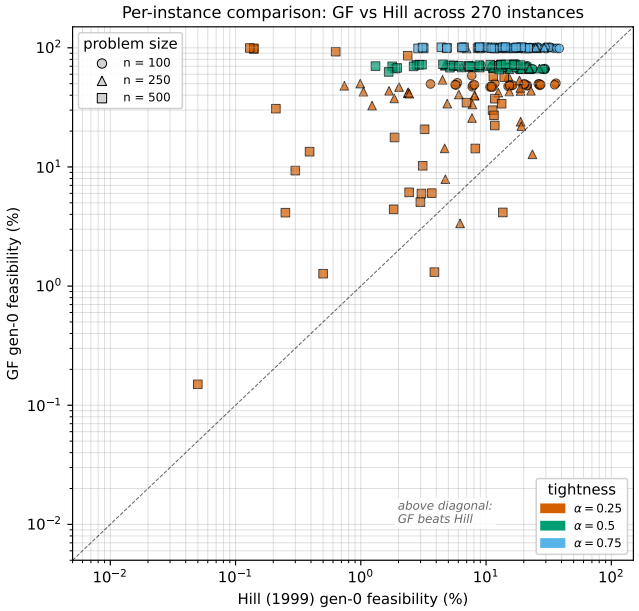

Figure 2. *Per-instance gen-0 feasibility comparison across 270 instances. Points above the diagonal indicate GF outperforms Hill. The 4 losses sit just below; see Section 6.4.*

Honest framing. The decoupling is statistical, not guaranteed. The 4 losses of Section 6.4 all occurred at $\text{ESS}_{\min} = 1$, confirming that low ESS does correlate with increased failure probability. Restart-on-collapse exists precisely because this decoupling can fail on individual seeds.

6.7 Adaptive q^* Distribution

The pilot consistently selects $q^* < \alpha$ at tight constraints, especially at large n:

Cell	$\alpha = 0.25$	$\alpha = 0.50$	$\alpha = 0.75$
mknapcb1 (100,5)	0.150	0.400	0.650
mknapcb2 (250,5)	0.250	0.550	0.650
mknapcb3 (500,5)	0.200	0.400	0.650
mknapcb4 (100,10)	0.300	0.400	0.650
mknapcb5 (250,10)	0.200	0.550	0.650
mknapcb6 (500,10)	0.200	0.550	0.650
mknapcb7 (100,30)	0.200	0.400	0.650
mknapcb8 (250,30)	0.150	0.550	0.650
mknapcb9 (500,30)	0.200	0.550	0.650

Table 10. *Pilot-selected q^* (mode across the 10 instances per cell). At tight constraints q^* falls 5–10 percentage points below α at large n.*

$q^* \neq \alpha$ universally—the pilot is doing real work and a fixed q would be suboptimal.

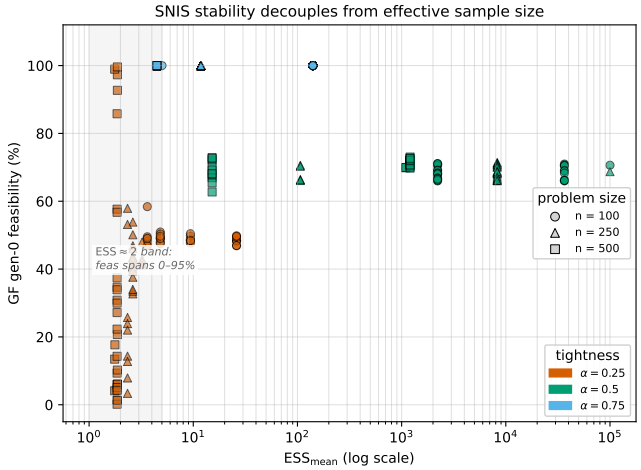

Figure 3. *Per-instance gen-0 feasibility plotted against effective sample size. At* ESS ≈ 2 *(the leftmost cluster, $n = 500, \alpha = 0.25$), feasibility outcomes span* 0–95%*, demonstrating that ESS undercounts the stability of the self-normalised ratio estimator. Restart-on-collapse handles the residual instability at the low-ESS low-feasibility tail.*

6.8 Population Size Sensitivity

Mean GF feasibility at $\alpha = 0.25$ across evaluation population sizes from 100 to 10,000 is approximately flat (variation under ± 3 percentage points within each cell): the gen-0 advantage holds at realistic GA population sizes (100–500), not just the 10,000-sample evaluation size. Full table in supplementary material.

7 Discussion

7.1 Phase Regimes Across α

The benchmark spans three constraint-tightness regimes, and the method exhibits qualitatively different behavior in each.

Tight ($\alpha = 0.25$). Uniform Bernoulli is catastrophic (0% feasibility everywhere); Hill gives 1–26%; GF gives 16–50% with restart firing on the hardest cells. IS is in the weight-degenerate regime; SNIS ratio remains stable on most instances. The regime where the contribution is largest.

Medium ($\alpha = 0.50$). Uniform 13–27%; Hill 2–27% (Hill underperforms because Toyoda packs heavy items aggressively); GF 66–71% uniformly across all 9 cells. GF wins on all 90 instances.

Loose ($\alpha = 0.75$). Uniform $\sim$ 100%; Hill underperforms ($\sim$ 17%) because dense value-aware sampling exceeds capacity; GF correctly degenerates to uniform. The theoretically correct fallback, not a failure mode.

7.2 Why GF Works Structurally

The Hill method computes a single global p reflecting average constraint tightness, then samples all items independently from Bernoulli(p). This loses the per-item structural information: items heavy across many constraints are sampled at the same rate as items that are light or constraint-irrelevant. The GF method's per-item $\hat{p}_j$ captures this structural information directly. An item heavy on multiple binding constraints has a lower $\hat{p}_j$ (feasibility-weighted solutions exclude it); an item light or constraint-irrelevant has a higher $\hat{p}_j$. The resulting population biases toward solutions that "make sense" given the constraint geometry.

At $\alpha = 0.75$, all items fit comfortably and the per-item structural signal vanishes—the GF method correctly identifies this and produces uniform $\hat{p}_j \approx 0.5$. This is not a failure mode but a correctness property: the framework respects the structure of the problem and does nothing when there is nothing to do.

7.3 Applicability Beyond Standard MKP

The framework's most important property is that it is *objective-free*: $\hat{p}_j$ depends only on weights w_{ji} and capacities W_i, not on values v_j. This makes it applicable to settings where repair operators are undefined or unreliable: constraint satisfaction problems (no objective function exists), noisy or uncertain objectives, and adversarial value-weight correlation structures where greedy efficiency rankings systematically mislead [8, 5]. Demonstrating these advantages empirically is a primary direction for follow-up work (Section 8.1).

7.4 Limitations

No downstream GA performance claims. This paper characterizes initialization quality at generation zero. We do not claim that improved gen-0 feasibility translates to faster GA convergence or better final solutions. Those claims require running GAs to convergence under controlled conditions and depend on choices—repair operator, fitness model, crossover/mutation rates, selection scheme—that interact with initialization in non-trivial ways. Under soft penalty (the more common fitness model for MKP) in particular, differences in initial population feasibility can be substantially compressed by the penalty mechanism over the course of evolution; we therefore make no claim about downstream Conv-90 or final-quality results.

ESS degeneracy at very large scales. The simple Bernoulli(q) proposal becomes weight-degenerate at $n \geq 500$ with tight constraints. SNIS partially compensates and restart-on-collapse catches edge cases, but the underlying issue is fundamental: extending to $n = 1000$ or beyond with the current proposal class is unlikely to scale gracefully. Sequential Monte Carlo with adaptive proposal refinement is the natural extension (Section 8.1).

Construction heuristics not directly comparable. Random feasible greedy, Sahni's algorithm, and surrogate constraint methods fuse initialization with a specific sequential repair operator and achieve 100% gen-0 feasibility by construction. A fair comparison would require specifying a downstream repair operator for the probabilistic methods, which is outside the scope of this paper. For completeness, random feasible greedy produces val_max approximately 2–5% above GF, but this comparison is not in the same class.

The 4 losses. Four instances at $\alpha = 0.25$ with $\text{ESS}_{\min} = 1$ lose to Hill, with gen-0 feasibility just above the 1% restart threshold. A threshold of 5% would have recovered all four; we report 1% as a conservative choice and identify threshold adaptation as future work.

Computational cost. GF-biased initialization is substantially more expensive per-instance than Hill or uniform Bernoulli. We do not report numerical timing comparisons in this paper: our GPU-accelerated implementation and the standard CPU-based Toyoda heuristic are not on the same hardware-software stack, and the cost is intentionally one-shot per instance, amortising across multiple GA generations or multi-seed benchmark trials. Formal timing analysis is deferred to follow-up work.

Order-1 approximation. The framework estimates per-item feasibility independently using order-1 schemas. For items with strong pairwise interactions, the order-1 approximation may over- or under-estimate the true per-item feasibility contribution. Extending to order-2 schemas is conceptually straightforward but multiplies computation by a factor of n.

Known optima unavailable. The OR-Library file format used here does not include known optimal values; we report val_max relative to the Hill baseline rather than as a percentage of optimum.

8 Conclusion and Future Work

We have presented a generating function-informed initialization framework for constrained binary optimization, grounded in the same algebraic tradition that Hardy and Ramanujan [9] used to analyze integer partitions and applied here to a new question: how much feasible space does each item's inclusion or exclusion preserve? By computing per-item feasibility densities from sub-generating functions and deriving per-bit sampling probabilities via proportional allocation, we obtain an initialization method that is principled, purely constraint-driven, and self-calibrating: it degenerates to uniform initialization exactly when no constraint structure is present.

The framework decomposes into two algorithmic phases. A simple Monte Carlo estimator with a Bernoulli(0.5) proposal is sufficient at small problem sizes; at scale, where the Bernoulli(0.5) proposal collapses, we develop a self-normalised importance sampling extension with adaptive proposal-parameter selection and a restart-on-collapse mechanism. On the full Chu–Beasley OR-Library benchmark (270 instances; n up to 500, m up to 30), the method beats the Hill (1999) baseline on 266 of 270 instances at generation zero, achieves 100% win-rate at medium and loose constraints and 95.6% at tight constraints, and characterizes the regime where the self-normalised ratio estimator remains stable despite low effective sample size.

8.1 Future Work

1. **Sequential Monte Carlo for very high dimensions.** The simple Bernoulli(q) proposal degenerates at $n \geq 500$. SMC with adaptive proposal refinement (tempered sequence from uniform to constraint-aware) is the natural extension; the current $\hat{p}_j$ vector can serve as the SMC initial distribution.

2. **Downstream GA convergence experiments.** Quantify whether improved gen-0 feasibility translates to faster GA convergence and better final solution quality under standard fitness models, with fixed hyperparameters across init methods and multiple seeds.

3. **Random feasible greedy probe as an alternative proposal.** The placeholder of Section 4.4 suggests three open experimental questions: per-item probe estimates versus SNIS $\hat{p}_j$; probe as a sequential IS *proposal* with weights computed against the corresponding proposal distribution; relative cost on GPU.

4. **Adaptive threshold selection.** The 1% collapse threshold was chosen heuristically. Adaptive selection based on pilot variance or per-instance characteristics could recover the four current losses.

5. **Adversarial and correlated instances.** Test instances with negative value-weight correlation [8] or strong item-weight correlations (Pisinger's hard knapsacks [5]) could demonstrate regimes where repair-based approaches degrade while constraint-only GF-biased initialization remains effective.

6. **Other problem classes.** Set covering, generalized assignment, multiple-choice knapsack, and constraint satisfaction problems where no objective is defined.

7. **Higher-order schemas.** Extending feasibility density estimation to order-2 schemas to capture pairwise item interactions.

8. **Theoretical SNIS variance bounds.** Concentration inequalities for the SNIS ratio under specific feasibility structures could replace the current empirical characterization of the SNIS–ESS decoupling with rigorous bounds.

Acknowledgments

The author thanks Professor Raymond R. Hill (Air Force Institute of Technology) for detailed correspondence on an earlier draft of this work, including specific methodological suggestions that motivated the importance sampling extension of Section 4.3 and the random feasible greedy probe of Section 4.4. Professor Hill also raised the question of fitness-model interactions with initialization that prompted the gen-0-only scope of the empirical evaluation. The author additionally thanks Ryan Bockman and Dr. Sandy Ganzell for collaboration on a related formalisation of the order-1 schema framework.

References

[1] P. C. Chu and J. E. Beasley, *A Genetic Algorithm for the Multidimensional Knapsack Problem*, Journal of Heuristics, 4(1):63–86, 1998.

[2] J. E. Beasley, *OR-Library: Distributing Test Problems by Electronic Mail*, Journal of the Operational Research Society, 41(11):1069–1072, 1990.

[3] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[4] H. S. Wilf, *generatingfunctionology*, 2nd edition, Academic Press, 1994.

[5] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*, Springer, Berlin, 2004.

[6] R. R. Hill, *A Monte Carlo Study of Genetic Algorithm Initial Population Generation Methods*, In Proc. 1999 Winter Simulation Conference, eds. P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, pp. 543–547, 1999.

[7] R. R. Hill and C. Hiremath, *Improving Genetic Algorithm Convergence Using Problem Structure and Domain Knowledge in Multidimensional Knapsack Problems*, International Journal of Operational Research, 1(1–2):145–159, 2005.

[8] R. R. Hill, J. T. Moore, C. Hiremath, and Y. K. Cho, *Test Problem Generation of Binary Knapsack Problem Variants and the Implications of their Use*, International Journal of Operations and Quantitative Management, 18(2):105–128, 2012.

[9] G. H. Hardy and S. Ramanujan, *Asymptotic Formulae in Combinatory Analysis*, Proceedings of the London Mathematical Society, 2(17):75–115, 1918.

[10] R. R. Hill, personal communication, May 2026.