# A Comparative Analysis of Approximation Algorithms for the 1DBPP

Grouping Genetic Algorithm (GGA)

Team - Alan Turing

AAD - Monsoon 2025

## 1 Meta Heuristic Approach for 1D Bin Packing

A metaheuristic is a high-level problem-solving framework designed to find a near-optimal solution for complex optimization problems, particularly those that are NP-hard, like the one-dimensional bin packing problem (1DBPP). Unlike a specific algorithm that follows a rigid set of instructions, a metaheuristic provides a general guiding strategy to explore the solution space effectively.It does not guarantee finding the single best solution, but it can generate high-quality solutions in a practical amount of time by guiding simpler heuristics. The Grouping Genetic Algorithm (GGA) is a prime example of a metaheuristic approach that empirically seeks near-optimal solutions by evolving groups of items to find better packing configurations.

Key characteristics of metaheuristics include:

- **No Formal Guarantees:** They do not provide a formal approximation ratio but are valued for their strong average-case performance on irregular or complex datasets.
- **Nature-Inspired:** Many metaheuristics are inspired by natural processes. Genetic algorithms, for instance, mimic the principles of biological evolution, such as selection, crossover, and mutation, to iteratively improve a population of potential solutions.
- **Balancing Search Strategies:** They strategically balance exploration (searching broadly across the entire solution space to find new possibilities) and exploitation (focusing on refining the best solutions found so far to make them even better).

## 2 Key Steps of the Grouping Genetic Algorithm (GGA)

The Grouping Genetic Algorithm (GGA) is a metaheuristic that mimics the process of natural selection to find a near-optimal solution for the bin packing problem. It works by iteratively refining a "population" of potential solutions over many "generations."

1. **Initialization:** The algorithm begins by creating an initial population of candidate solutions. For the bin packing problem, each solution is a complete, valid arrangement of all items into a set of bins. These first-generation solutions are typically generated randomly and are often inefficient.

2. **Fitness Evaluation:** Each solution in the population is evaluated and assigned a fitness score. The primary goal is to minimize the number of bins used. Therefore, a solution using fewer bins is considered "fitter" and receives a better score. Secondary metrics, such as how full the bins are, can also be used to break ties.

3. **Selection:** The "survival of the fittest" principle is applied. The best solutions—those with the highest fitness scores—are selected to be "parents" for the next generation. This ensures that the characteristics of high-quality packings are carried forward.

4. **Crossover:** This is the core step where solutions are combined to evolve better groupings of items. Two parent solutions are selected, and their genetic material is mixed to create a new "child" solution. In the GGA context, this often involves taking a group of well-packed bins from one parent and combining them with bins from the other parent, then intelligently re-organizing any leftover or duplicated items.

5. **Mutation:** To ensure genetic diversity and prevent the algorithm from getting stuck on a non-optimal solution, small, random changes are introduced into the child solutions. A typical mutation for bin packing might involve randomly moving a single item to a different bin or swapping two items between bins.

6. **Termination:** The cycle of evaluation, selection, crossover, and mutation is repeated for many generations. The algorithm stops when a termination condition is met, such as reaching a maximum number of generations, or when the quality of the best solution has not improved for a set number of iterations. The best solution found throughout this entire process is then presented as the final answer.

# 3 Chromosome Representation and Grouping

In the Grouping Genetic Algorithm (GGA), the representation of a solution (the **chromosome**) is uniquely designed to solve grouping problems like bin packing. Instead of focusing on individual items, its entire structure is built around the concept of the **group**.

Choosing an efficient representation like this is arguably the most critical design decision in a genetic algorithm. The representation fundamentally defines the **search space** the algorithm will explore. A naive representation, such as a simple list of all items in order, creates a search space so vast and full of invalid or non-sensical solutions that finding a good one becomes computationally infeasible. A "smart" representation, like the grouping approach, effectively prunes this space, ensuring that the algorithm only explores solutions that are structurally meaningful, making the search far more efficient.

Furthermore, the representation dictates the very shape of the **fitness landscape**. A well-designed chromosome creates a "smoother" landscape, where solutions with similar fitness are "close" to each other. This allows the genetic operators, like crossover, to make intelligent, large-scale changes that move the search toward better regions. Conversely, a poor representation creates a rugged and chaotic landscape riddled with "local minima"—traps where the algorithm gets stuck on a mediocre solution, unable to find the nearby path that leads to a much better, globally optimal one.

# 4 A couple of Approaches to GGA for 1D BPP

## 4.1 A Hybrid Grouping Genetic Algorithm for 1D BPP

EMANUEL FALKENAUER

### Chromosome Representation and Encoding

In this specific approach, the chromosome is encoded to make the **groups** the fundamental "genes" of the solution. The chromosome itself is composed of two parts: a **group part** and a **standard item part**. The group part is the primary component that the genetic operators will work with. This part is a variable-length list, where each element represents a single group (i.e., one bin).

The standard item part of the chromosome merely serves as a reference, identifying which specific items actually form each group defined in the group part. This ensures that every item is accounted for exactly once in a valid solution. This encoding is critical because it forces the genetic algorithm to operate on the level of entire bins, which are the meaningful building blocks of a bin packing solution. This allows the algorithm to effectively exploit and combine good packing structures found in parent solutions.

### Example of Encoding

The **standard item part** is a fixed list of all items, while the **group part** is the chromosome that evolves. The chromosome's value at a certain position tells you which group the item at that same position belongs to.

| **Standard Item Part (Fixed)** | **Group Part (Chromosome)** |
|---|---|
| Item A (0.6) | Group 1 |
| Item B (0.5) | Group 2 |
| Item C (0.4) | Group 1 |
| Item D (0.3) | Group 3 |
| Item E (0.2) | Group 2 |

Based on this, the final solution is decoded as:

- **Group 1 (Bin 1)**: {Item A, Item C}
- **Group 2 (Bin 2)**: {Item B, Item E}
- **Group 3 (Bin 3)**: {Item D}

### The Crossover Operation

The GGA crossover operation works with variable-length chromosomes where the genes represent the groups. Given the fact that the hard constraints and the cost function vary among different grouping problems, the ways groups can be combined without producing invalid or "too bad" individuals are not the same for all those problems. Thus, the crossover used will not be the same for all of them.

However, it will fit the following pattern:

1. Select at random two crossing sites, delimiting the crossing section, in each of the two parents.
2. Inject the contents of the crossing section of the first parent at the first crossing site of the second parent. Recall that the crossover works with the group part of the chromosome, so this means injecting some of the groups from the first parent into the second.
3. Handle item duplication. Items from the injected section that now appear twice in the child chromosome are identified. The algorithm then **eliminates the entire old groups (bins)** that these duplicate items were members of in the second parent. This ensures the old membership gives way to the new, injected group structure and maintains a valid solution.
4. **Re-insert items with a local heuristic.** The items that were part of the *eliminated old groups* (from step 3) are now unassigned. A local, problem-dependent optimization heuristic is then applied to intelligently insert this list of removed items back into the child chromosome, finding the most efficient way to add them to existing or new groups (bins).
5. Apply points 2 through 4 to the two parents with their roles permuted in order to generate the second child.

As can easily be seen, the idea behind the GGA crossover is to promote **promising groups by inheritance**. We wish to get solutions that inherit the best packing patterns from the parent solutions/chromosomes.

### The Mutation Operation

The purpose of mutation is to maintain genetic diversity in the population and prevent the algorithm from converging prematurely to a local optimum. It introduces small, random changes to a child chromosome after the crossover operation.

In the context of the Grouping Genetic Algorithm, a common mutation strategy is to select one or more groups (bins) at random from the chromosome. These selected groups are then dissolved, and all their constituent items are added to the list of unassigned items. Finally, the same local heuristic used in the crossover step is applied to re-insert these items back into the solution, finding the most efficient placement for them either in other existing groups or by forming new ones.

### The Dominance Criterion

This algorithm uses a local optimization heuristic based on the **dominance criterion** proposed by Martello and Toth. The core idea is simple: a bin $B_2$ is said to "dominate" bin $B_1$ if the items in $B_1$ can be partitioned in a way that they are "no bigger" than the items in $B_2$. This means a solution using $B_2$ is guaranteed to be no worse than a solution using $B_1$.

Finding the absolute best, most dominant bin is an exponential-time problem. To make this procedure computationally practical, the paper uses an approximation where the search for dominance is limited to sets of **three or less items**. This $n = 3$ approximation is not the main algorithm itself, but rather serves as a powerful local optimization heuristic. It is used to produce high-quality, dominant bins that are then efficiently processed by the Grouping Genetic Algorithm.

### The Cost Function (Problem Redefinition)

A simple cost function, like minimizing the number of bins, is unusable in practice. It creates an "extremely unfriendly landscape" where a vast number of slightly suboptimal solutions (e.g., $OPT + 1$ bins) all have the same cost. This "needle in a haystack" problem provides no guidance for the algorithm to follow.

To solve this, the paper redefines the problem as **maximizing** the following cost function:

$$f_{BPP} = \frac{\sum_{i=1}^{N} \left(\frac{F_i}{C}\right)^k}{N}$$

Where $N$ is the number of bins in the solution, $F_i$ is the total fill (sum of item sizes) of bin $i$, $C$ is the bin capacity, and $k$ is a constant, $k > 1$.

The constant $k$ is used to reward "extremist" or very well-filled bins. The larger the $k$, the more the algorithm prefers a few "elite" bins over many equally filled ones. The paper finds that $k = 2$ gives good results, as larger values can lead to premature convergence on local optima. Crucially, for $k \leq 2$, this function $f_{BPP}$ is shown to have the same global optima as the original problem, ensuring that $f_{BPP}(P_{N+1}) < f_{BPP}(P_N)$ (a solution with $N+1$ bins will always score lower than an optimal solution with $N$ bins).

### Summary of Steps

The complete Grouping Genetic Algorithm (GGA) follows this iterative process:

1. **Initialization:** Generate an initial population of random, valid solutions (chromosomes).
2. **Fitness Evaluation:** Calculate the fitness of each solution in the population using the defined cost function $f_{BPP}$.
3. **Selection:** Select parent solutions from the population based on their high fitness scores.
4. **Crossover:** Apply the grouping crossover operation to the selected parents to create new child solutions. This process includes:
   - Injecting "group" sections from one parent to another.
   - Eliminating entire old groups that contain duplicate items.
   - Using a local heuristic (like the $n = 3$ dominance criterion) to re-insert the unassigned items efficiently.
5. **Mutation:** Apply the mutation operator to the new child solutions (e.g., dissolve a few random groups and re-insert their items using the same local heuristic) to maintain diversity.
6. **New Population:** The newly created child solutions form the next generation's population.
7. **Termination:** Repeat steps 2-6 until a termination condition is met (e.g., a maximum number of generations or no improvement in the best solution). The best-scoring solution found during the entire run is the final answer.

## 4.2 Another Grouping Genetic Algorithm for 1D BPP

HITOSHI IIMA AND TETSUYA YAKAWA

### Key Differences and Techniques

This paper proposes another GA design that, while also emphasizing the "combination of items in a bin" as the key factor, differs significantly in its representation and genetic operators.

- **Chromosome Representation:** The chromosome (genotype) is a direct sequence of item sets, where each set represents one bin. This is a key distinction, as the authors note that the solution is not dependent on the *order* of the genes (bins) in the chromosome, which contrasts with the previous approach.
- **Generation of Initial Solutions** In a typical genetic algorithm, the first generation (initial population) is often created completely at random. However, to start the search process with better-quality solutions, this approach uses a heuristic.

  Instead of generating solutions randomly, the algorithm creates each initial solution by applying the **First-Fit (FF)** procedure. To ensure the initial population is diverse, the items are sequenced **at random** before the FF procedure is applied to them. This creates a variety of different, reasonably good solutions to form the starting population.

- **Complex Crossover:** The crossover operation is highly specialized. It first copies compatible, non-conflicting bins from both parents into the offspring[cite: 187]. Remaining items from partially-used bins are put into a temporary set, which is then split into single-item sets ($T_a$) and multi-item sets ($T_b$). The algorithm then attempts to improve the offspring's bins using a "Replacement procedure" with items from $T_a$ and $T_b$.
- **Hybrid Heuristic Re-insertion:** This approach explicitly introduces heuristic methods *into* the crossover and mutation operations. After the replacement step, any items still in the temporary sets are re-inserted using a two-phase procedure: first, the **First-Fit (FF)** heuristic is used to pack them into existing bins, followed by the **MBS'** heuristic to pack the rest into new bins. This is a different local optimization strategy than the dominance criterion.
- **Mutation:** The mutation operator follows a similar pattern. It randomly selects and dissolves two or three bins, placing all their items individually into a temporary set. It then applies the *exact same* replacement and two-phase (FF/MBS') re-insertion procedures that are used in the crossover step.
- **Selection Strategy:** The algorithm explicitly uses the **Minimal Generation Gap (MGG)** model for its selection operation.

### The Minimum Bin Slack (MBS) Heuristic

The Minimum Bin Slack (MBS) heuristic, or its variant MBS', is a key component used as a local optimization tool within this GA. Unlike item-focused heuristics (like FF), MBS is a **bin-focused** method. Its main goal is to create a new bin and pack it as tightly as possible, minimizing the "slack" (unused capacity).

- It first sorts all available items in decreasing order of size.
- It then uses a recursive **"one-packing-search"** procedure to find the best possible combination of items that can fit into a single new bin.
- This procedure is repeated to create more bins until all items are packed.

In this paper's GA, a variant called **MBS'** is used during the crossover and mutation operations. After items are removed and must be re-inserted, MBS' is the powerful heuristic used to intelligently pack these remaining items into new bins, ensuring the new child solution is both valid and high-quality.

# 5 Expected Outcomes for GGA Analysis

Based on the project proposal, the analysis of the Grouping Genetic Algorithm will contribute to the overall project with the following specific outcomes: