Homework 3

GRASP

Advanced Algorithm

Supervisor:

Dr. Ziarati

Atefe Rajabi

40230563

Spring 2024

Index

Abstract:

This report explores the application of the Greedy Randomized Adaptive Search Procedures (GRASP) algorithm to the one-dimensional bin packing problem (1D-BPP), a classical combinatorial optimization problem characterized by its NP-hard status. This project evaluates the algorithm's performance with various local search strategies, including Hill Climbing, Tabu Search, and Simulated Annealing, and discusses the impact of key parameters such as the alpha value and heuristic probability on the effectiveness and efficiency of the solution. Experimental results, visualized through plots and detailed in comparative analyses, demonstrate the potential of GRASP to find near-optimal solutions effectively.

Introduction:

The bin packing problem (BPP) poses significant challenges in fields ranging from logistics and manufacturing to data storage and distribution. Given a set of items and bins, the task is to optimally pack the items such that the number of bins used is minimized without exceeding the capacity of any bin. This problem is inherently NP-hard, making exact solutions computationally impractical for large datasets. Heuristic and metaheuristic approaches are therefore critical in deriving efficient solutions. Among these, the Greedy Randomized Adaptive Search Procedures (GRASP) algorithm presents a promising avenue due to its adaptive and iterative nature, which allows for the exploration of a diverse set of potential solutions. This report delves into the application of GRASP to the 1D-BPP, focusing on its configuration, execution, and the role of various algorithmic parameters in influencing its performance. The effectiveness of GRASP is assessed through a series of experiments involving different local search strategies, providing insights into their comparative advantages and the overall efficacy of the algorithm in reducing the number of bins utilized while optimizing computational resources.

Bin Packing problem:

The 1D bin packing problem (1D-BPP) is a classical combinatorial optimization problem that deals with efficiently packing a finite set of items into bins. Each item has a weight (or size), and each bin has a maximum capacity that cannot be exceeded. The goal is to minimize the number of bins required to pack all the items. This problem is an example of an NP-hard problem, meaning it is computationally challenging to find an exact solution, especially as the size of the problem grows.

Here are the key components of the 1D bin packing problem:

1. Items: Each item has a specific size or weight.

2. Bins: All bins have the same maximum capacity, and no bin can hold items exceeding this capacity.

3. Objective: The primary objective is to minimize the number of bins used to pack all the items.

This problem has practical applications in various industries, such as logistics for optimizing space in cargo containers, in computing for data storage optimization, and in manufacturing for minimizing material waste.

The challenge lies in how to arrange the items in the bins in such a way that the number of bins is minimized, considering the capacity limit of each bin. Various algorithms, especially heuristic and metaheuristic algorithms, are employed to find near-optimal solutions to this problem due to its complexity and the impracticality of solving it exactly for large instances.

Solution Feasibility:

If both checks are passed—no bins are overloaded and the total weight of items in the bins matches the total weight of all items—the function returns ‘true’, confirming that the solution is feasible. This means:

- All items are accounted for without duplication.

- No bin exceeds its capacity.

This feasibility check is crucial for ensuring that any proposed solution to the bin packing problem adheres to the fundamental constraints of the problem, thereby confirming that the solution is viable and correct.

Objective Function:

The objective function proposed in the paper[1] focuses on optimizing the placement of items in bins by considering not only the number of bins used but also the amount of unused space within these bins. Specifically, to avoid stagnation and encourage the algorithm towards finding optimal solutions, the objective function incorporates the amount of unused space alongside the count of bins.

Minimization Objective Function: This function aims to minimize the inverse of the sum of the filled capacity ratios of all bins. Mathematically, it is represented as minimizing

$$1 - \sum(fill_k/C)^z/N$$

, where $fill_k$ is the filled capacity of bin k, C is the fixed bin capacity, z is a constant used to define the equilibrium of the filled bin, and N represents the number of bins in the solution.

These objective function prioritizes solutions that use fewer bins and minimize the unused space within those bins, aiming for efficient bin utilization. The inclusion of z, typically set to 2, helps balance the emphasis on bin count versus the filled capacity, guiding the optimization towards more desirable solutions.

The constant z in the objective functions of the optimization problem is a significant parameter that influences how the solution's quality is assessed in terms of both bin utilization and the number of bins used. Here's a detailed explanation:

Role of z

The constant z plays a crucial role in balancing these two aspects. Specifically, it adjusts how much emphasis is placed on the filled capacity of the bins relative to their total capacity.

Mathematical Interpretation

The exponent z modifies the impact of the ratio $fill_k/C$ on the objective function:

[1] Munien, C., & Ezugwu, A. E. (2021). Metaheuristic algorithms for one-dimensional bin-packing problems: A survey of recent advances and applications. *Journal of Intelligent Systems*, 30, 636-663. https://doi.org/10.1515/jisys-2020-0117

- When $z > 1$, the function places increasing emphasis on bins that are closer to being full. This means that as bins get filled closer to their capacity, they contribute more significantly to the objective function, encouraging solutions that efficiently utilize bin space.

- When $z = 1$, the function simplifies to a linear relationship with the fill ratio. In this case, each percentage point increase in bin utilization is valued equally, regardless of how full the bin is.

- When $z < 1$, the function would, hypothetically, place less emphasis on bins being fully utilized.

GRASP Algorithm:

The GRASP (Greedy Randomized Adaptive Search Procedures) algorithm is a heuristic method used to solve combinatorial optimization problems. It operates iteratively, combining elements of greediness and randomness to efficiently find high-quality solutions.

1. Phases of GRASP: Each iteration of GRASP consists of two main phases:

- Construction Phase: Starts with an empty solution and iteratively adds elements to build a feasible solution. The choice of elements is guided by a greedy function, which is adapted after each selection to reflect the changing state of the solution space. This phase uses a restricted candidate list (RCL), which limits the choices to the most promising candidates based on the greedy function.

- Local Search Phase: Once a feasible solution is constructed, the local search phase attempts to improve this solution by exploring its neighborhood. This process continues until no further improvements can be found.

2. Adaptive and Randomized Nature: The adaptive nature of the algorithm comes from updating the greedy function based on the current solution's context, allowing the algorithm to adjust its strategy dynamically. The randomized component stems from randomly selecting elements from the RCL during the construction phase, which helps in escaping local optima and exploring diverse solution spaces.

3. Iterative Process and Stopping Criteria: GRASP repeats its two-phase process multiple times. Each iteration starts with a fresh construction of the solution, potentially leading to different starting points for the local search. The overall process continues until a predefined stopping criterion is met, such as a time limit, convergence threshold or a satisfactory solution quality.

Parameters:

Each parameter plays a specific role in influencing the behavior and performance of the algorithm. Let's go through each parameter and understand its significance:

1. 'z':

- The constant z in the objective functions of the optimization problem is a significant parameter that influences how the solution's quality is assessed in terms of both bin utilization and the number of bins used.

2. 'max_iteration':

- The maximum number of iterations the algorithm will execute before stopping. This acts as a hard cap on the number of complete cycles the algorithm will run, providing a safeguard against infinite loops and ensuring termination even if the optimal solution is not necessarily found.

3. 'no_improvement_threshold':

- This parameter sets the number of consecutive iterations without any improvement in the solution after which the algorithm will terminate. It helps in avoiding excessive computation time when further iterations do not yield better results, thereby optimizing computational efficiency.

4. 'stopType':

- Indicates the criterion used to stop the algorithm. For instance:
  - '0' suggests stopping after a fixed amount of real-time has elapsed.
  - '1' represents stopping after a fixed number of iterations.
  - '2' represents stopping after no improvement in solutions after certain times.

5. 'time_limit':

- Specifies the maximum amount of time (in seconds) that the algorithm is allowed to run.

6. 'heuristic_probability':

- This parameter controls the probability of choosing between different heuristic strategies during the solution construction phase. For example:

- A probability (e.g., 0.3) indicates the likelihood of choosing the 'First Fit' heuristic versus the 'Best Fit' heuristic in a bin-packing.

7. ‘alpha’:

- The alpha parameter in GRASP controls the size of the Restricted Candidate List (RCL) by defining the threshold of how far from the best current solution a candidate can be to still be considered for inclusion in the RCL, effectively balancing between exploration and exploitation in the search process.

8. ‘localSearch’:

- Specifies the type of local search procedure used to refine each solution found during the constructive phase of GRASP. The options are:

- ‘0’ for Hill Climbing: A straightforward local search method that moves to neighboring solutions that are better than the current one until no improvements can be found.

- ‘1’ for Tabu Search: An advanced local search technique that uses a memory structure to avoid cycling back to previously visited solutions, allowing it to potentially escape local optima.

- ‘2’ for Simulated Annealing: Uses a probabilistic technique to make decisions about whether to accept worse solutions in the hope of escaping local optima, with the acceptance probability decreasing over time.

GRASP Parameters value:

{ "z": 2,
"max_iteration": 3000000,
"no_improvement_threshold": 40,
"stopType": 0,
"time_limit": 40,
"heuristic_probability": 0.3,
"alpha": 0.5,
"localSearch": 1}

Hill Climbing Parameters value:

{ "z": 2,
"max_iteration": 3000000,
"no_improvement_threshold": 40,
"stopType": 0,
"time_limit": 20,
"tweaksCount": 30}

Tabu Search Parameters value:

{ "z": 2,
"max_iteration": 300000,
"no_improvement_threshold": 10,
"stopType": 0,
"convergence_threshold": 5.0,
"time_limit": 20,
"tabuSize":10,
"tweaksCount": 5}

Simulated Annealing Parameters value:


```
{
  "z": 2,
  "max_iteration": 10000,
  "schedule": 2,
  "no_improvement_threshold": 50,
  "acceptance_rate": 0.90,
  "stopType": 0,
  "no_overall_improvement_threshold": 20,
  "time_limit": 10
}
```

Implemented Algorithm:

1. Configuration and Initialization

- Configuration Loading: Depending on the type of local search strategy (Hill Climbing, Tabu Search, or Simulated Annealing), different configuration files are loaded using the ‘read_localConfig’ function, which reads settings such as no improvement threshold, stop type, etc., from specified configuration paths.

2. Solution Construction

- Shuffling Items: Items are shuffled in ‘shuffle_items’ to ensure a varied starting point for each iteration, which helps in exploring different regions of the solution space.

- Construction Phase: During the ‘construction_phase’, a new solution is built by iteratively placing items into bins. The selection of bins for items uses the Restricted Candidate List (RCL), which is influenced by the alpha parameter. The heuristic probability determines whether to use a 'first-fit' or 'best-fit' approach for placing the items.

3. Local Search Integration

- Local Search Calls: After each construction phase, a local search is applied based on the selected strategy (Hill Climbing, Tabu Search, or Simulated Annealing). Each strategy is encapsulated in its respective method (‘HC_phase’, ‘TS_phase’, ‘SA_phase’), which applies specific improvements based on the local search paradigm.

- Local Search Configuration: Each local search method uses configurations loaded previously and applies these during the search to enhance the current solution iteratively.

4. Adaptive Behavior

- Parameter Adaptation: The method ‘adapt_parameters’ is designed to modify the alpha and heuristic parameters if no improvement is seen over a certain number of iterations, increasing the exploration capability dynamically.

5. Stopping Conditions

- Multiple Stopping Criteria: The algorithm can stop based on time (‘stopBy_time’), a maximum number of iterations (‘stopBy_iteration’), or a convergence criterion (‘stopBy_Convergence’) which checks if there has been no improvement over a set number of iterations.

- Time Management: Execution time for each phase and overall is tracked and used as part of the stopping conditions.

8. Algorithm Execution Loop

- The main loop (‘algorithm’) manages the iterative process of solution construction and local search, checking stopping conditions, and adjusting parameters based on the algorithm's performance.

Neighborhood Definition:

Here's the implemented tweak functions:

1. Single Item Move: This Tweak attempts to move an item from one bin to another that has sufficient capacity. The item and bin are selected randomly.

2. Swap: This Tweak attempts to swap two items between two different bins. The swap only occurs if both bins have sufficient capacity to accommodate the swapped items. This could help in optimizing space usage across bins.

3. Merge Bins: This function tries to merge the contents of one bin into another if there's enough capacity. This could reduce the total number of bins used, thereby optimizing the solution.

4. Empty Bin: This function randomly selects a bin and attempts to redistribute all its contents to other bins with sufficient capacity, aiming to entirely empty the selected bin and possibly reduce the total number of bins used.

5. Item reinsertion: This function selects an item at random and moves it to a newly created bin, potentially altering the bin configuration and item distribution to find a better packing solution.

Among implemented Tweaks, Only Empty Bin and Swap have been used.

Reasons:

Adaptive Selection Mechanism[2]

The adaptive selection mechanism uses rewards and penalties to guide the selection of tweaks during the optimization process. The method is designed to prefer tweaks that have historically led to improvements and to avoid those that frequently result in worse solutions. Here's how it works:

1. Initialization and Updating of Probabilities:

[2] Inspired by "Xue, Y., Zhu, H., Liang, J., & Słowik, A. (2021). Adaptive crossover operator based multi-objective binary genetic algorithm for feature selection in classification. *Knowledge-Based Systems, 227*, 107218. https://doi.org/10.1016/j.knosys.2021.107218"

- Each tweak type is assigned a probability of being selected, which is initially set equally across all tweaks.

- After each application of a tweak, rewards and penalties are updated based on the tweak's effectiveness. Rewards are given for improvements in the solution, and penalties are given for non-improvements or worsening of the solution.

2. Reassign Probabilities:

- The probabilities of selecting each tweak are recalculated based on the recent history of rewards and penalties.

- This is done by computing a value for each tweak using the formula: $Value = \frac{rewards}{rewards+penalties}$ - These values are then normalized to ensure they sum to one, forming a new set of probabilities that guide the future selection of tweaks.

- If a probability is zero (indicating consistent failure of a tweak), it is adjusted to a small positive value to ensure that no tweak is ever completely excluded, allowing for exploration.

3. Selection of Tweak:

- When a tweak is to be selected, a random value is generated and used to select a tweak probabilistically based on the cumulative distribution of the tweak probabilities.

- This selection is inherently adaptive as it favors tweaks that have been more successful but still allows for exploration of less successful tweaks.

Construction Phase:

The construction phase of the GRASP (Greedy Randomized Adaptive Search Procedures) algorithm for a bin packing problem incorporates several crucial components essential for operation and effectiveness. Here's a breakdown of the primary elements and their roles within the construction phase:

1. Data Initialization and Configuration

- Item Shuffling: The ‘shuffle_items’ function is employed to randomly shuffle the order of items. This randomization helps in preventing the algorithm from being trapped in local optima by starting each iteration with a different sequence of items.

2. Construction of the Solution

- Iterative Item Placement: Each item is placed into bins iteratively using a decision process defined by the Restricted Candidate List (RCL). For each item, possible bins (candidate bins) are determined based on the space available and the bin's compatibility with the item's properties.

- RCL Determination and Selection:

- The RCL for placing each item into a bin is determined using the ‘RCL_maker’ function, which can choose between two heuristics (‘first_fit’ and ‘best_fit’) based on a probabilistic decision influenced by the ‘heuristic_probability’ and a random threshold.

- The ‘alpha’ parameter adjusts the size of the RCL by truncating it to a certain proportion of its original size, allowing for a balance between greediness and randomness in the selection process.

- A bin from the RCL is randomly selected for placing the current item.

3. Bin Management

- Bin Initialization and Update: If the selected bin from the RCL already exists, the item is added to it. If not, a new bin is created with a new identifier, and the item is added.

- Capacity Management: The remaining capacity of each bin is adjusted upon the addition of new items, ensuring that the constraints of the bin packing problem are adhered to throughout the construction phase.

The construction phase efficiently integrates randomized and greedy decision-making to construct initial solutions for the bin packing problem, effectively balancing between exploration of the

solution space and exploitation of good packing configurations. This sets the stage for potentially more intensive local search optimizations in subsequent phases of the algorithm.

Heuristics:

In the context of bin packing algorithms like GRASP, the Best Fit and First Fit heuristics are crucial for efficiently placing items into bins with the aim of minimizing the total number of bins used or maximizing the use of available space within bins. These heuristics help generate candidate bins for each item during the construction phase of the algorithm. Here's a detailed explanation of how each heuristic operates and how they contribute to generating candidate lists for the construction phase:

Best Fit Heuristic

- Definition: The Best Fit heuristic places each new item into the bin that will leave the least amount of leftover room after the item is placed, among the bins that have enough remaining capacity to accommodate the item.

- Operation:

- For each item, the heuristic scans through all existing bins and calculates the remaining capacity for each bin if the item were to be added.

- It selects the bin where the item fits most snugly – i.e., the bin that will have the smallest remaining capacity after placing the item but still enough to accommodate the item without exceeding the bin's limit.

- If no existing bin can accommodate the item, a new bin is created for it.

- Application in GRASP:

- In the ‘best_fit’ method, the function begins from a randomly selected starting index and checks subsequent bins for available capacity to accommodate the current item. Bins that have enough remaining capacity are added to the candidate list. This approach, while incorporating elements of random selection, helps in identifying multiple suitable bins rather than focusing on the single tightest fit, potentially leading to diverse placement strategies and efficient space utilization across various iterations of the algorithm.

First Fit Heuristic

- Definition: The First Fit heuristic places each new item into the first bin that has enough space for the item, scanning bins in the order they were created or based on some other fixed sequence.

- Operation:

- For each item, the bins are checked in a sequential order, and the item is placed into the first bin that has sufficient capacity to hold the item.

- If no bin can accommodate the item, a new bin is opened for it.

- Application in GRASP:

- In the ‘first_fit’ method, bins are checked in the sequence until a suitable bin is found. This heuristic gives more candidates compared to Best Fit heuristic so it introduces more randomness in the search procedure.

Integration in GRASP's RCL (Restricted Candidate List)

- Role in Candidate Generation:

- The RCL is used in GRASP to introduce a degree of randomness into the selection process, which helps explore a wider solution space and potentially escape local optima.

- The choice between First Fit and Best Fit heuristics for generating the RCL can be randomized or conditioned on certain probabilities (‘heuristic_probability’), allowing the algorithm to switch dynamically between focusing on space efficiency (Best Fit) and computational efficiency (First Fit).

- Adjustment by ‘alpha’ Parameter:

- After generating a list of candidate bins using either heuristic, the list may be truncated based on the ‘alpha’ parameter, which controls the size of the RCL. This truncation typically involves keeping only a certain top percentage (as defined by ‘alpha’) of bins that best fit the criteria (e.g., smallest remaining capacity for Best Fit).

Local Search:

In our implemented GRASP algorithm, we have integrated three distinct local search algorithms—Hill Climbing, Tabu Search, and Simulated Annealing—as part of an experimental exploration to evaluate their efficacy when combined with heuristic methods. Each local search algorithm offers unique exploration and exploitation strategies, contributing to the algorithm's ability to refine solutions obtained during the construction phase. Hill Climbing focuses on iteratively improving solutions by making small incremental changes, while Tabu Search introduces memory-based search mechanisms to navigate through the solution space effectively. Simulated Annealing, inspired by the annealing process in metallurgy, introduces probabilistic transitions to facilitate exploration and escape local optima. By integrating these diverse local search strategies, our aim is to empirically assess their performance in enhancing the GRASP algorithm's ability to find high-quality solutions across a variety of problem instances, ultimately guiding us towards the selection of the most effective optimization approach for our specific problem domain.

Libraries:

In the implementation of the GRASP algorithm for the bin packing problem, several external libraries have been utilized to enhance functionality and streamline various operations. Matplotlibcpp is employed for plotting and visualization, enabling a graphical representation of the algorithm's performance over different iterations and configurations. This visualization aids in understanding the optimization trends and the effectiveness of different parameter settings. The Json library is used to handle configuration files, allowing for flexible and easily adjustable parameters that can be modified without altering the core codebase. This capability is crucial for experimenting with different algorithm settings to determine optimal configurations. Lastly, Libxl is integrated for its capabilities in reading from and writing to Excel files, facilitating easy data manipulation and storage. This is particularly useful for documenting the results in a structured format that is suitable for analysis and presentation, thereby supporting extensive evaluation and reporting of the algorithm's outcomes. These libraries collectively contribute to a robust development environment that supports efficient data handling, result visualization, and configuration management, crucial for comprehensive optimization studies.

Results:

All the plots, visualization, outputs are available in the project folder.

These results are derived from 10 algorithm execution with the presented configuration.

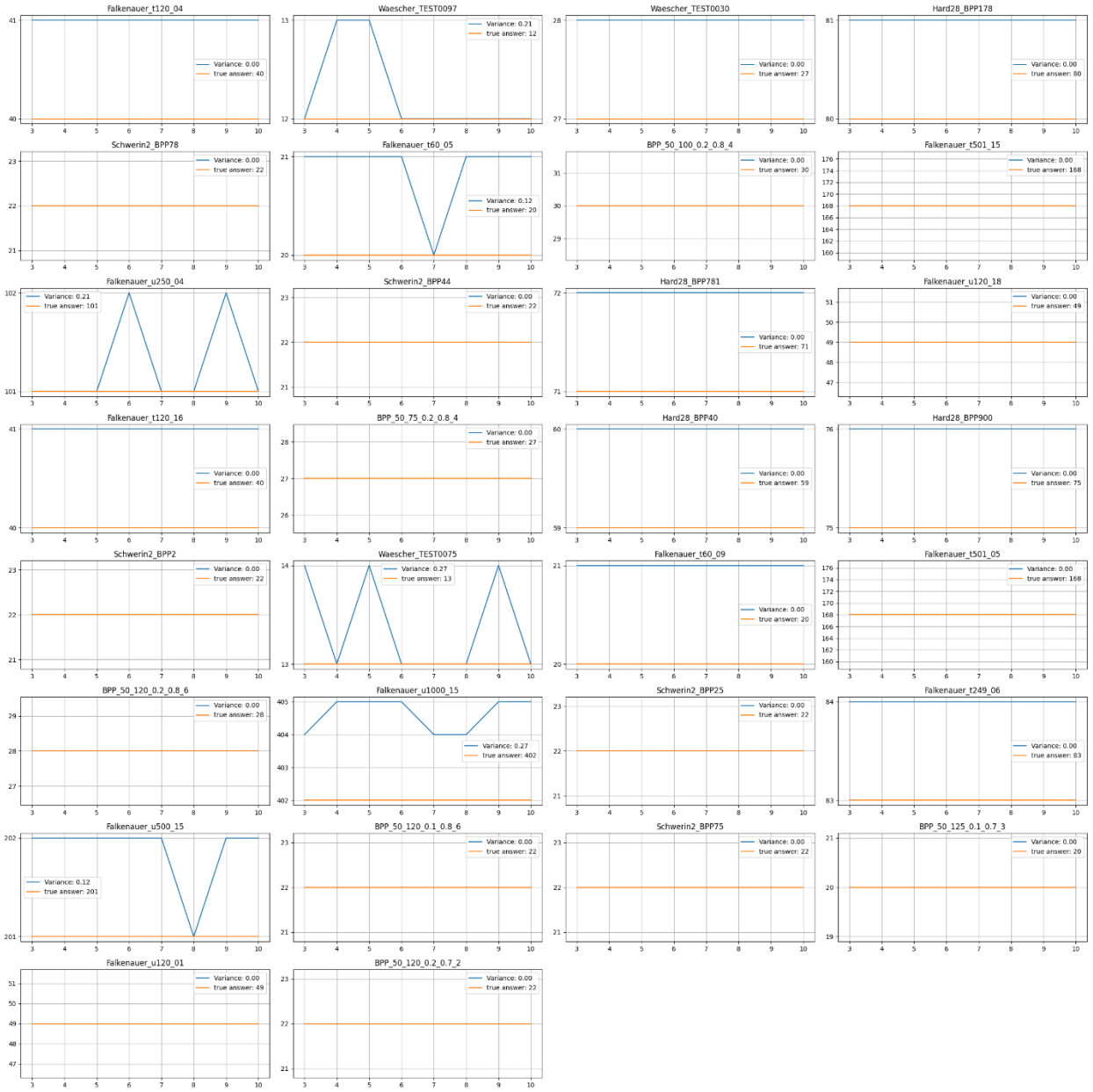
Instance Set	Optimum solution	solution Hill Climbing (best – worst)	solution Tabu Search (best – worst)	solution Simulated Annealing (best – worst)	Average time (Hill Climbing)
Falkenauer_t120_04	40	41	41	41	1
Waescher_TEST0097	12	12 - 13	12 - 13	12 - 13	0
Waescher_TEST0030	27	28	28	28	0
Hard28_BPP178	80	81	82	82	3
Schwerin2_BPP78	22	22	22	22	0
Falkenauer_t60_05	20	20 - 21	20 - 21	20 - 21	0
BPP_50_100_0.2_0.8_4	30	30	30	30	0
Falkenauer_t501_15	168	168	168	168	19
Falkenauer_u250_04	101	101 - 102	101 - 102	101 - 102	8
Schwerin2_BPP44	22	22	22	22	0
Hard28_BPP781	71	72	72	72	6
Falkenauer_u120_18	49	49	49	49	0
Falkenauer_t120_16	40	41	41	41	0
BPP_50_75_0.2_0.8_4	27	27	27	27	0
Hard28_BPP40	59	60	60	60	2
Hard28_BPP900	75	76	76	76	3
Schwerin2_BPP2	22	22	22	22	1
Waescher_TEST0075	13	13 - 14	13 - 14	13 - 14	1
Falkenauer_t60_09	20	21	21	21	0
Falkenauer_t501_05	168	168	168	168	19
BPP_50_120_0.2_0.8_6	28	28	28	28	0
Falkenauer_u1000_15	402	404 - 405	404 - 405	404 - 405	19
Schwerin2_BPP25	22	22	22	22	1
Falkenauer_t249_06	83	84	84	84	10
Falkenauer_u500_15	201	201 - 202	201 - 202	201 - 202	19
BPP_50_120_0.1_0.8_6	22	22	22	22	0
Schwerin2_BPP75	22	22	22	22	1
BPP_50_125_0.1_0.7_3	20	20	20	20	0
Falkenauer_u120_01	49	49	49	49	1
BPP_50_120_0.2_0.7_2	22	22	22	22	0

Variance:

Orange line: Optimum solution

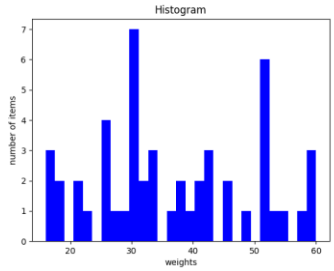
Blue line: Solution with GRASP Algorithm (Hill Climbing Local Search)

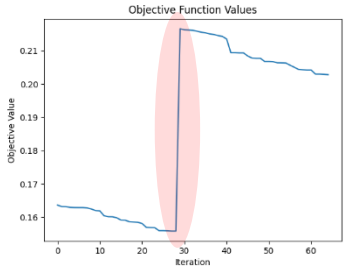
Based on objective function plots, it is obvious that Some instance sets need more time to converge.



Plots: (Hill Climbing Local Search)

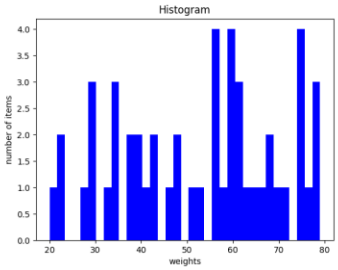
\BPP_50_75_0.2_0.8_4

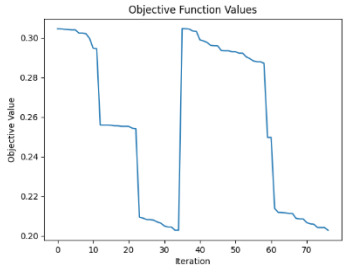




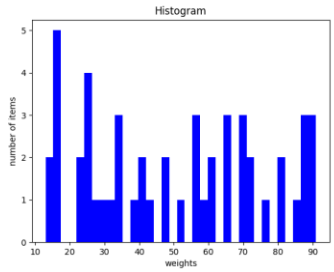
The increase in the objective value indicates that the GRASP algorithm initiated a new start. However, the subsequent decrease in other parts of the objective value suggests improvements

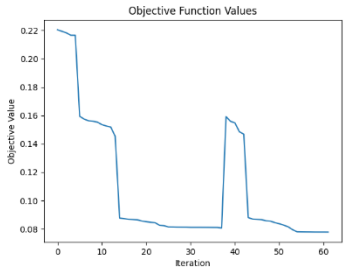
\BPP_50_100_0.2_0.8_4



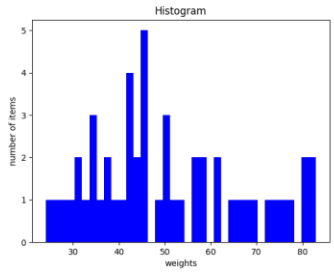


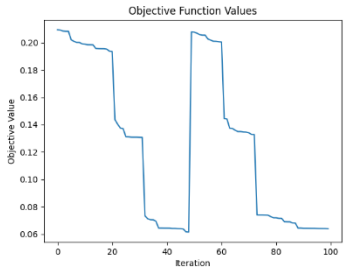
\BPP_50_120_0.1_0.8_6



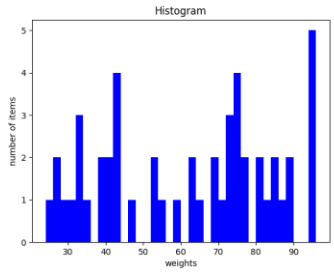


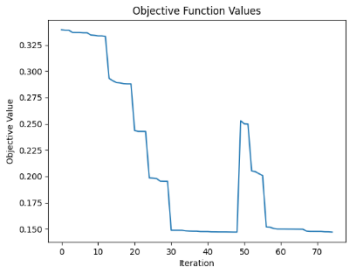
\BPP_50_120_0.2_0.7_2



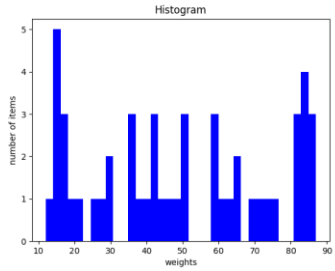


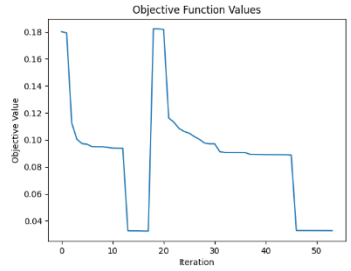
\BPP_50_120_0.2_0.8_6



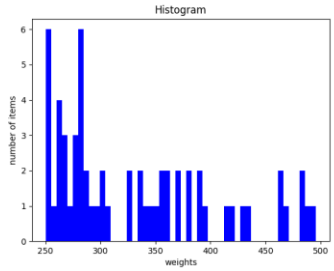


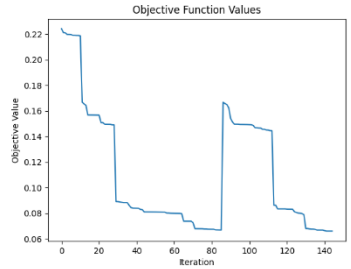
\BPP_50_125_0.1_0.7_3



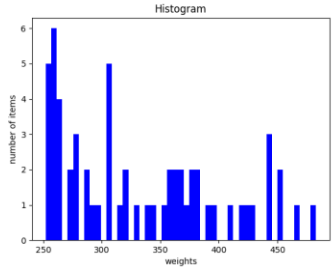


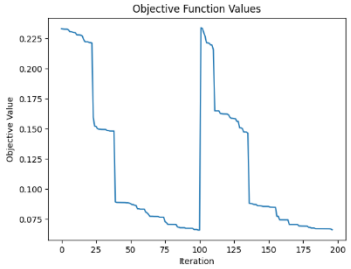
\Falkenauer_t60_05



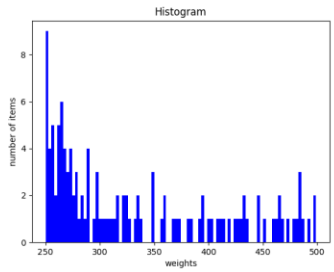


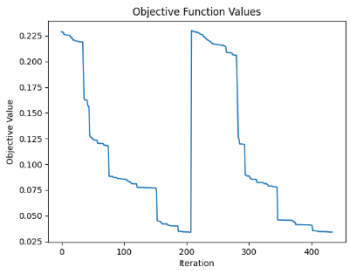
\Falkenauer_t60_09



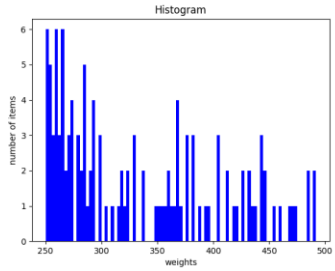


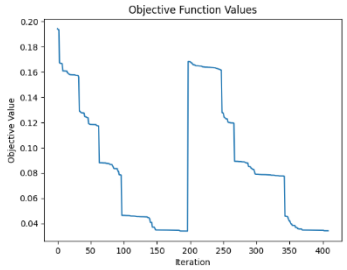
\Falkenauer_t120_04



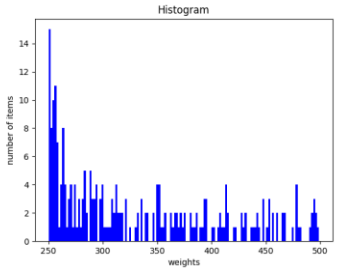


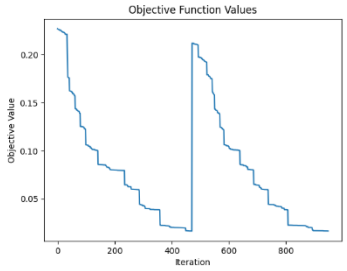
\Falkenauer_t120_16



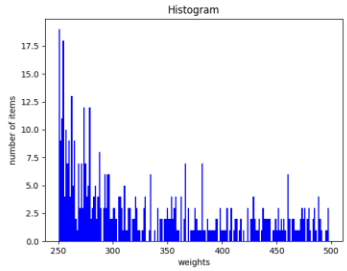


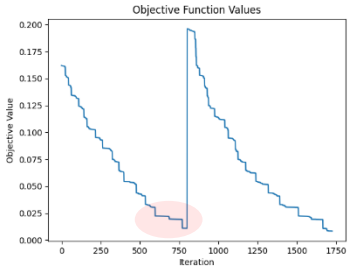
\Falkenauer_t249_06





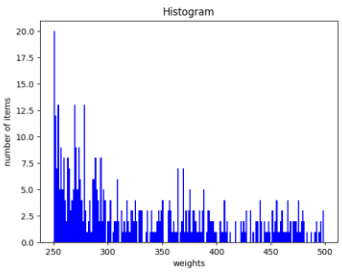
\Falkenauer_t501_05

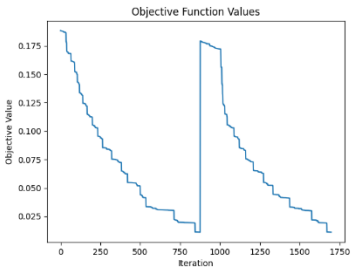




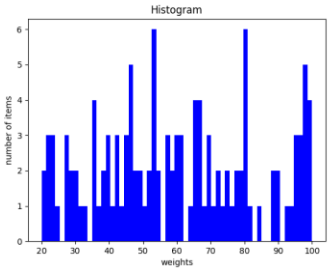
\Falkenauer_t501_15

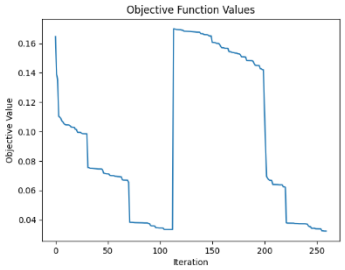
Based on the decreasing trend observed in the plot, it is clear that the local search phase required more time to converge to an optimal value for this complex set of instances, as indicated by the data distribution.



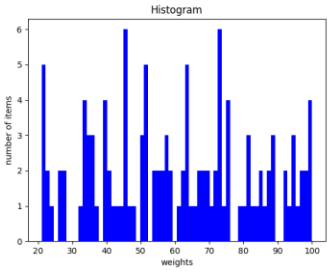


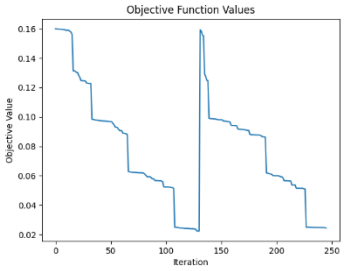
\Falkenauer_u120_01



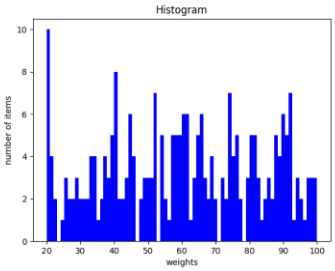


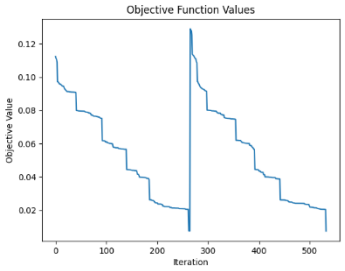
\Falkenauer_u120_18



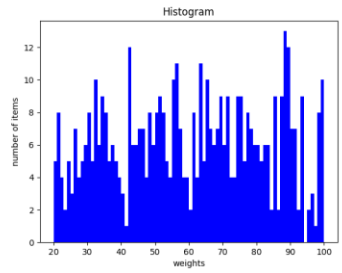


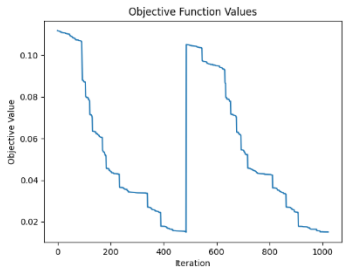
\Falkenauer_u250_04



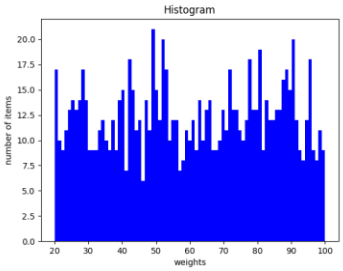


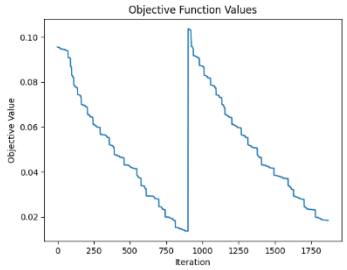
\Falkenauer_u500_15



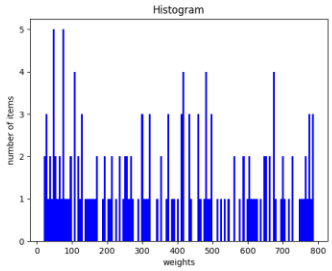


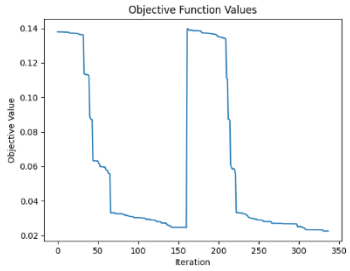
\Falkenauer_u1000_15



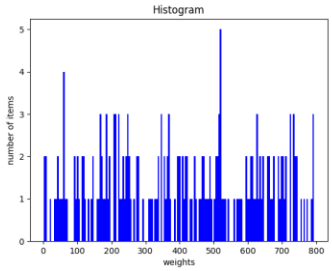


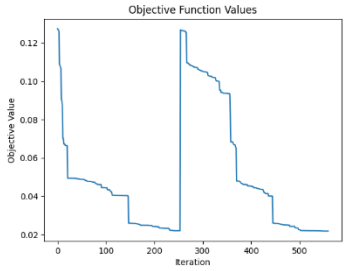
\Hard28_BPP40



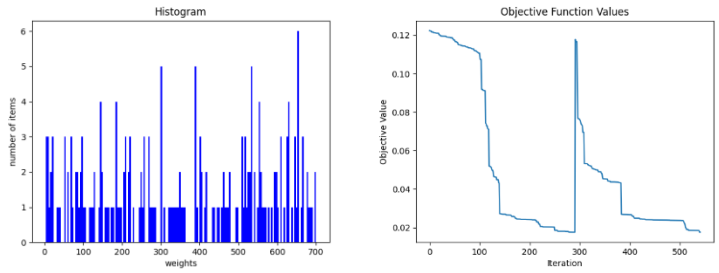


\Hard28_BPP178

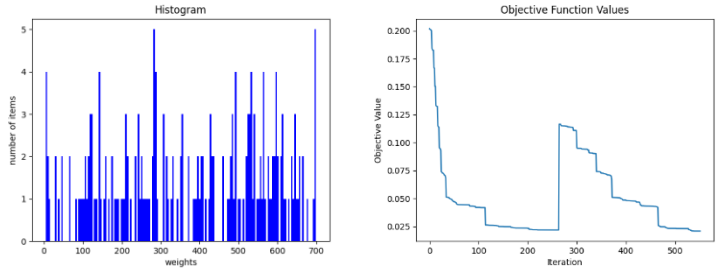




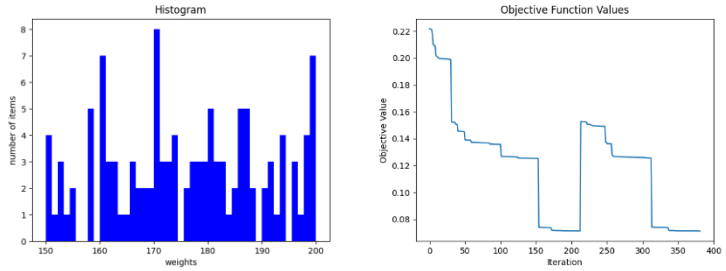
\Hard28_BPP781



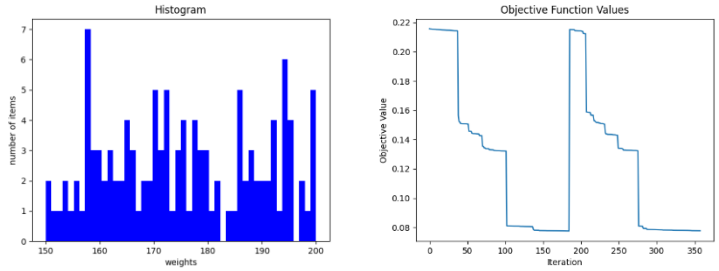
\Hard28_BPP900



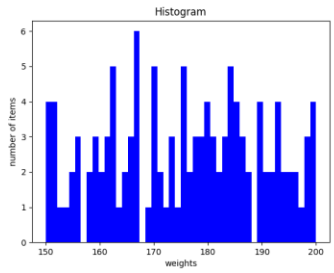
\Schwerin2_BPP2

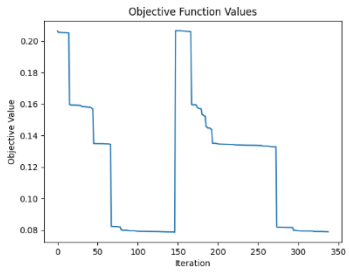


\Schwerin2_BPP25

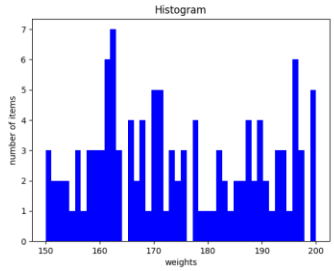


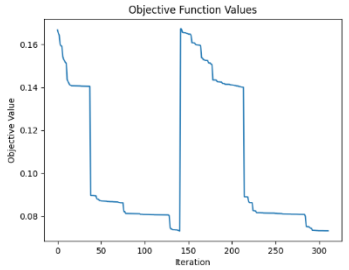
\Schwerin2_BPP44



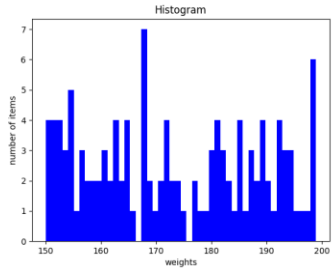


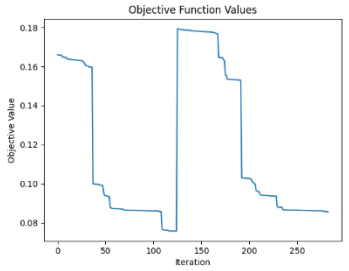
\Schwerin2_BPP75



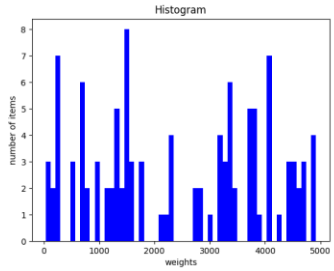


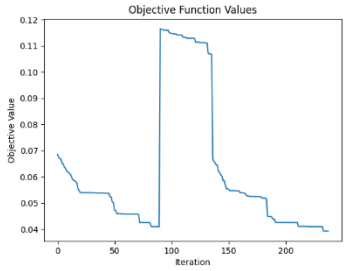
\Schwerin2_BPP78



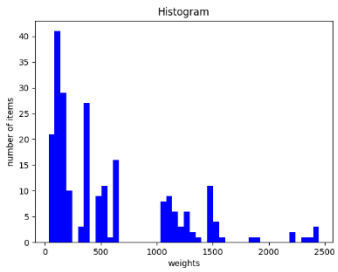


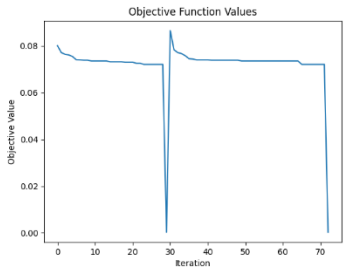
\Waescher_TEST0030



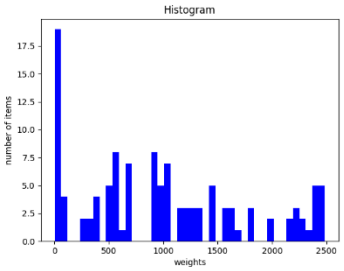


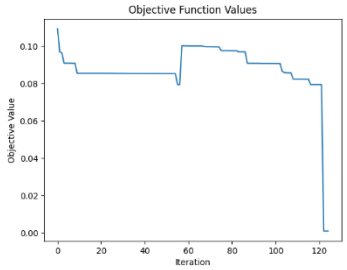
\Waescher_TEST0075





\Waescher_TEST0097





Discussion:

1. Converge Faster and Reduce Search Time: GRASP aims to converge towards high-quality solutions quickly by iteratively improving the solutions obtained during the construction phase through local search procedures. By focusing on promising regions of the search space and iteratively refining solutions, GRASP can reduce the overall search time compared to exhaustive search methods.

2. Introducing Randomness: The randomized element in GRASP's solution construction phase allows for the exploration of a broader range of candidate solutions. This randomness prevents the algorithm from getting stuck in local optima and encourages the discovery of diverse and potentially better solutions.

3. Avoiding Convergence to Suboptimal Solutions: While GRASP aims to converge quickly, it also strives to avoid premature convergence to suboptimal solutions. By introducing randomness, GRASP ensures that the algorithm continues to explore the search space, even after finding locally optimal solutions. This helps prevent the algorithm from prematurely terminating the search before potentially better solutions are discovered.

4**.** Reduced Variance in Solutions: Unlike other non-greedy randomized algorithms, GRASP tends to exhibit less variance in its solutions. This consistency arises from its structured approach to solution generation and refinement. By combining greedy construction with randomization and rigorous local search, GRASP consistently produces solutions that are not only high-quality but also display less variability between different runs of the algorithm.

5. Tuning parameters: Since the GRASP algorithm itself does not have as many parameters as other metaheuristics, tuning is much simpler, and the algorithm is more robust to noise in the parameters compared to others.

Alpha:

The alpha parameter in GRASP controls the balance between exploitation and exploration during the solution construction phase. A smaller alpha (in rank based approach: alpha * RCL_size) promotes more exploitation, favoring greedier choices and potentially leading to quicker convergence but with a higher risk of getting stuck in suboptimal solutions. In contrast, a larger alpha encourages more exploration by allowing for a broader range of randomized choices, which can help escape local optima but may require more iterations to converge to high-quality solutions.

Heuristic probability:

If the best fit heuristic is more probable during the construction phase, it means that the algorithm is biased towards selecting bins that minimize the remaining capacity the most. Consequently, using the best fit heuristic tends to result in a higher number of bins being created compared to using the first fit heuristic. This is because the best fit heuristic tends to create new bins more frequently in order to accommodate items more closely, leading to a finer-grained packing arrangement with potentially more bins overall.

Shuffle items:

Shuffling items before the construction phase in GRASP introduces randomness into the algorithm by changing the starting point for constructing each solution. This randomness increases the exploration of the search space, as the algorithm is exposed to different initial configurations of items. By exploring a more diverse set of starting points, GRASP can potentially discover different solutions and avoid being biased towards a specific initial arrangement of items. This increased exploration can help the algorithm escape local optima and find higher-quality solutions.

Local Search:

Without incorporating an exploratory local search algorithm, such as simulated annealing or tabu search, GRASP may indeed struggle to escape local optima, potentially leading to suboptimal solutions.