Bin-Packing with Genetic Algorithm Approach and WoC

Abdurrahman AlQuran & Obadah Abozaid

Neuroscience/CSE & CSE

Speed School of Engineering

University of Louisville, USA

asalqu02@louisville.edu & osaboz01@louisville.edu

***Abstract* - The bin-packing problem is a well-known NP-Complete problem that involves finding the minimum number of bins of a fixed capacity that can fit a set of items of varying sizes. This paper presents a hybrid algorithm that combines a genetic algorithm (GA) with a wisdom of crowds (WoC) approach to solve the bin-packing problem. The GA is a bio-inspired optimization technique that mimics the process of natural evolution, while the WoC is a collective decision-making method that aggregates multiple independent solutions. The paper also introduces a greedy refinement function that rearranges the items in the bins to ensure their validity and optimality. The paper evaluates the performance of the hybrid algorithm on different instances of the bin-packing problem and compares it with other existing algorithms. The results show that the hybrid algorithm can find near-optimal solutions in a reasonable time, and that the WoC and the greedy refinement can improve the quality of the solutions. The paper also discusses the advantages and limitations of the hybrid algorithm and suggests some directions for future work.**

Index Terms - Bin Completion algorithm, Bin packing problem, Crossover, Fitness function, Genetic Algorithm, Mutation, Tournament selection, Wisdom of Crowds

I. INTRODUCTION

FOR THIS final project implementation of an AI algorithm, we will be making use of our lovely genetic algorithm (GA), an evolutionary approach [1], while also using the Wisdom of Crowds (WoC) method from the previous project. This time around though, we will be creating this implementation on an NP-Complete problem of our choosing. For this project's purposes, my group partner and I chose the bin-packing problem [2]. An innovation of the popular knapsack problem, which turned out to be not allowed for this project despite being NP-Complete (since our previous classes no doubt have covered this problem, so it would not be too interesting), the bin-packing problem involves the incorporation of various sized objects into bins all a set size. For our intents and purposes, all items are between size 0 and 1, so x is of size $0 \leq x \leq 1$, and items will fall within bins all of size 1.00. The sum of the objects within a bin, otherwise the total filled capacity of a bin, should not exceed a value of 1. Additionally, a bin that is currently filled to a size of exactly 1.00 is completely filled, and thus should not be filled with any additional objects. On the contrary, a bin that has a value of 0.00 would be completely empty.

For this project, we revolutionized the GA structure we had utilized for the traveling-salesperson problem [3]. On top of the already plentiful variables at play, such as mutation rate, crossover rate, the population size, tournament size, and the target distance, this time we would be re-employing the use of WoC. In the framing of our bin packing problem, our evolutionary approach will make use of a series of similarly structured tournaments to determine the best combinations of values [4]. Then, the WoC portion will observe the best bin results of each run of the GA, 10 times, and bias towards using those combinations of bins for the purpose of creating a better, more advantageous population. This function would allow the facilitation of the creation of populations with ideal

fitness, using the overlap of common bins in the most fit cases. The goal of fitness in this case would be to minimize the number of bins filled. Though as seen in the experiment prior, WoC would sometimes lean towards a less effective solution compared to a plain single instance of GA, especially at lower item counts. Thus, this function should be most effective or most improving at higher item counts, as we will see in various data sizes tested seen below. Additionally, it would be interesting to compare it to a solution deemed most completely optimal according to our research.

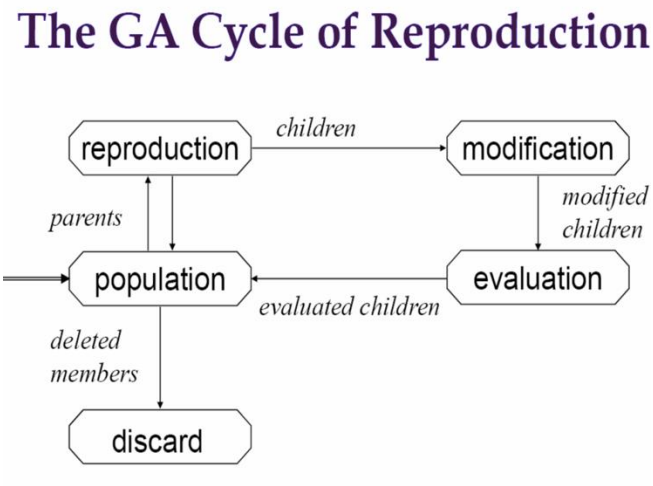


Fig. 1. A detailed flowchart of the general process of the genetic algorithm.

Unlike previous projects, where TSP files provided contained coordinates for a list of cities, our input would consist of a list of items of the various sizes from 0 to 1, and our output of packed bins would be fixed and arbitrarily chosen to constitute the dataset. Additionally, the bins are in the format of a 2D array, rather than a plain text file, such as the TSP plain text file type. We defined a fitness function which in this case evaluates the fitness of a chromosome, which is the negative of the bin overpacking penalty. Finally, we had to iron out any issues, leading to optimizations and fixes of the algorithm.

To approach the GA search heuristic, we used the WoC sorting method. This iteration of the GA will be another test of an evolutionary search approach to solve this NP-Complete problem. Albeit the GA will not create the most optimal path each time, and neither will it be identical each time as well as the random chance of mutation, but it will lead to a *favorable* solution for a logistics problem [5] through the evolutionary approach.

II. APPROACH

The python program for Bin-Packing with Genetic Algorithm Approach and WoC is as follows:

The program defines a bin-packing problem as finding the minimum number of bins of capacity 1.00 that can fit a list of items, varying in size. It uses a GA to find one of many solutions to the bin-packing problem. A GA is a biology-inspired optimization technique that mimics the process of natural evolution. The program also uses a novel WoC approach – as mentioned – to improve the solution quality [6], [7]. A WoC approach is based on the idea that the collective opinion of a group of individuals is better than that of a single 'expert.' The program consists of several functions that implement the GA and WoC steps. Here is a brief description of each function:

- `check_items(items)`: This function checks if the items list is valid (not empty and contains only numeric values).
- `initialize_population(num_items, population_size, max_bins)`: This function creates a random initial population of chromosomes, where each chromosome is a list of bin assignments for each item. The number of bins is limited by the max_bins parameter, a helpful metric for ensuring that bins stay within a reasonable magnitude [2].
- `fitness(chromosome, items)`: This function evaluates the fitness of a chromosome, which is the negative of the overpacked penalty. The overpacked penalty is the sum of the excess amounts in each bin that exceed the capacity of 1. The lower the penalty, the better the solution.
- `tournament_selection(population, fitnesses, k)`: This function performs a tournament selection of parents from the population, based on their fitness values. A tournament selection randomly chooses k individuals from the population and selects the best one as a parent. This is repeated until the number of parents is equal to the population size.
- `crossover(parent1, parent2)`: This function performs a single-point crossover between two parents, resulting in two offspring. A single-point crossover randomly chooses a point in the chromosome and swaps the segments after that point between the parents.
- `mutate(chromosome, max_bins, mutation_rate)`: This function performs a mutation on a chromosome, which randomly changes

some of the bin assignments with a given probability (`mutation_rate`). The mutation rate decreases by 0.99 every time a mutation occurs, to reduce the exploration over time.

- `genetic_algorithm(items, population_size, generations, max_bins)`: This function runs the GA for a given number of generations, using the previous functions. It returns the best solution found by the GA.
- `solution_to_bins(solution, items)`: This function converts a solution (a list of bin assignments) to a dictionary of bins, where each key is a bin index, and each value is a list of items in that bin.
- `run_woc(items, population_size, generations, max_bins, num_solutions)`: This function runs the WoC approach, which consists of generating `num_solutions` independent solutions using the GA, and then aggregating them to form a crowd solution. The aggregation is done by taking the most common bin assignment for each item among the solutions.
- `main()`: This function obviously defines the parameters and the items list for the bin-packing problem and calls the `run_woc` function to get the crowd solution. It also prints the bins and their sums, the total time, and the number of bins used by the crowd solution.

The Wisdom of Crowds (WoC) method is finally used to aggregate the solutions in the final population into a single solution [8]. The number of items and the total size of the items in each bin are also printed.

To refine the solutions provided by the WoC method and ensure their validity, we created a cleaning greedy algorithm that rearranges the items in the bins. This algorithm will ensure that each item is placed exactly once and that all bins have a total filled value of less than or equal to 1.00, since each bin could have various amounts following the renewal of the population from the crowds.

This function would sort items by size, in descending order of their sizes. This helps with efficiently filling the bins. It would then initialize a list of bins to add any needed bins to any displaced items. Then, we would place items in bins by iterating through each item and try to place it in an existing bin without exceeding the capacity (1.00). If no suitable bin is found, we would need to create a new bin. This is known as First filling, and though it is not entirely an optimal solution, it is very robust for our purposes.

Finally, the greedy function would need to validate and adjust: After the initial placement, we would check each bin to ensure it does not exceed the capacity. If any bin exceeds its capacity, it will then attempt to rearrange the items among other bins or create a new bin if necessary.

In our implementation, `greedy_refinement` takes the items and the WoC solution as input. It first sorts the items by size and then attempts to place them in bins based on the WoC solution. If any bin exceeds the capacity, it tries to rearrange the items among other bins or creates a new bin if necessary. The final bins are then returned as a list of lists, where it can then be fed into the print functions we created. Lastly, the time elapsed during the execution of the algorithm is measured and printed.

In terms of a genetic algorithm on its own, this implementation for the bin packing problem makes a proper use of the transmission model attributed to the key evolutionary features of the genetic tournament model. [9]. Thus, the way the problem effortlessly fits into the GA heuristic/schema further cements its role as a viable candidate for testing with such an algorithm.

Compared to previous implementation of the GA with WoC, we only introduce a new pair of functions which would identify common bin sequences within the most fit populations of bin combinations, and then creating a new population based on the best fit runs after more runs on higher counts of items I noticed a decent improvement with the addition of more runs of the GA, however we landed on 10 runs as a nice middle ground between runtime and improvement.

Additionally, to help clean up the solution provided by the WoC, we implemented a greedy function following the WoC to ensure proper bin-packing rules were followed the creation of a new population following WoC [10]. This would help drastically improve the quality of the data, since it was common for the WoC results to contain repeated or invalid solutions with bins of invalid amounts, meaning greater than 1.

```python
def greedy_refinement(items, woc_solution):
    sorted_items = sorted([(size, idx) for idx, size in enumerate(items)], reverse=True)
    bins = [[] for _ in range(max(woc_solution) + 1)]

    # Initial placement based on WoC solution
    for size, idx in sorted_items:
        bin_idx = woc_solution[idx]
        bins[bin_idx].append((size, idx))

    # Validate and adjust bins
    for bin in bins:
        if sum(size for size, _ in bin) > 1.0:
            # Attempt to rearrange items
            for size, idx in bin[:]:
                placed = False
                for other_bin in bins:
                    if other_bin is not bin and sum(s for s, _ in other_bin) + size <= 1.0:
                        other_bin.append((size, idx))
                        bin.remove((size, idx))
                        placed = True
                        break
                if not placed:
                    # Create a new bin if rearrangement is not possible
                    bins.append([(size, idx)])
                    bin.remove((size, idx))
```

Fig. 2. A snippet of code for refining function post WoC execution.

We decided we did not want to use a premade function from libraries such as NetworkX, available for python for functions such as the BFS (Breadth First Search) and DFS (Depth First Search) algorithms; instead these methods were created from scratch. My approach utilized some techniques from class. We settled on two functions for WoC: the first one creates a new population based on the most common bins from the winning, most fit outputs. From there, the new population would advance to the second, final function added, which was separate from the WoC: the greedy method.

To briefly discuss the GA methods, we would have a list of parameters which would be fed into the algorithm. These being the number of generations ran, the size of the population generated with each generation, the size of the tournament when comparing the best fit children, the rate of mutation and changing of values within the remaining children, the crossover rate of parents with preferential traits when creating children, and the preferred target distance before plateauing. Together, these all factor into how long and efficiently the algorithm will run. By adding more mutations, we can introduce more variation into the new population. By increasing crossover, we include more of the sequence most fit from the parents into the children. The population size will allow us to look at many potential best-fit sequences of items in bins within the same generation. Consequently, more generations mean a higher chance of narrowing down our options to the most fit. The algorithm will take two sequences in our population, crossover their data into children, mutate those children, then replace them into the population if they win the tournament. This process continues and loops.

We then used WoC, from which was adapted from the previous project, which takes out the *most common bins* from the 1000 generations of each of the 10 runs of the GA. This would then be used to create an informed population from which a new set of bins would be formulated, and a new tournament conducted. This function is not perfect, however, so another function would be needed as previously explained.

Further on the final function applied after the WoC: since the new population formation occurred as one final, culminating step at the end, there seemed to be no need for any greedy implementation to iron out any inconsistencies with the bin-packing solution guidelines. However, it was realized with testing that there were some mistakes being overlooked, so we wrote a new function that prunes away any repeat cities as well as which ensures every item is within a bin that was less than the designated maximum value.

Compared to other approaches of the bin packing problem, which may use more mathematically derived means of reorienting items within bins, this method of a tournament generation provides an interesting approach to a problem of absolute structure, where one most optimal solution exists and is dependent on the previous choices made in terms of previous arrangement of items in bins [11], [12], [13].

III. RESULTS

Our hybrid algorithm makes use of a GA which will choose the best parent sequences based on natural variation, and crossover of two parents to create two optimally more ideal children. We learned from our testing with the TSP problem that GA alongside WoC was one of the most efficient methods used to solve the TSP [14], as it resembles a more educated brute force or randomized approach, running in parallel and worked much more efficiently on larger scales [15], contributing to a polynomial time rather than factorial time. It was, however, not always as optimal or as quick as some greedy methods could get. Despite this, it was far faster than a completely unguided, random approach [16].

The total runtime of this approach can be approximated in big-O notation, as: O(number of tournaments * number of generations * population size * size of children). This leads to a CONSIDERABLY faster approach than brute forcing, though it may be rivaled by the greedy approach, since no high runtime operations such as

generating permutations, are necessary, though randomization and comparing, and crossover is taxing.

The program was run 5 separate times, then the data averaged. Since it is in the hundreds of milliseconds, temporal resolution was not of concern, so we wanted to ensure that we could get an average time for the program. Since our program did not make use of any GUI (Graphical User Interface), this was simple to run. A GUI would be helpful; however, it isn't really perfect due to the abstract nature of the bins. Despite this, we created one which would display each bin and its contents as a bar chart.

IV. DATA

Temporal resolution was no concern this time around since by design, the algorithm runs multiple instances, the crowd of sorts, before picking a final population to craft the winner. I ran 3 trials for various amounts of GA trials to see the marginal benefit of each number of tournaments. Thus, time is given as averages in seconds, and below are the results without two-opt applied whatsoever. The following is the comparison chart for GA vs. (GA & WoC) on same problems in terms of performance, speed, and optimality.

TABLE I

ALGORITHM PERFORMANCE

Bin sorting instance	**GA + WoC Fitness values**			**Regular GA Fitness values**		
10 items	Average bins	Best bins	Time (s)	Average bins	Best bins	Time (s)
10 runs	10	8	7.964	8	5	0.780
20 runs	9	8	15.582			
30 runs	7	6	23.510			
20 items						
10 runs	14	14	8.855	15	10	0.923
20	14	13	17.708			
30	12	11	26.559			
30 items						
5 runs [a]	18	17	5.104	19	17	1.041
10	18	17	11.323			
15	17	17	17.777			
40 item [b]						
5 runs [a]	31	28	6.741	29	26	1.239
10	30	27	12.482			
15	28	26	19.102			
50 cities						
3 runs [a]	36	33	4.126	35	32	1.314
6	35	33	8.161			
10	33	32	13.846			

Fitness in number of bins used and time for both WoC and non-WoC runs.

[a]We ran the number of items at various runs that scaled relative to the size of the number of items tested. [b]For the most part, the maximum number of bins was set to 20 for the first three sets. For the set of 40 and 50 items however, it was doubled, increased to 40 bins.

Bins that were over 1.00 were deemed unfit to include in the new population, and thus were skipped. Having many generations drastically helps the WoC. Without, we would commonly see 30-50% of the bins being unfit before the greedy heuristic. With higher item counts, it was more common for the program to fail with just the genetic algorithm and give invalid bin combinations. Or, it would underestimate the number of bins allotted, far under the maximum number allowed. With WoC however, it was found to consistently output the most fit solution, using the least number of bins possible, even with only 10 runs on item counts of 40 or 50. Other literature suggests the benefits of utilizing WoC at far larger scales than what was tested [17]. It was feared, however, that using a large enough crowd would create adverse effects for runtime, such that it defeated the purpose of running fewer generations within each genetic algorithm.

Yet, the WoC algorithm still completed in a timely fashion. Thus, 10 runs of the GA tournament for the WOC function were determined to be the sweet spot between optimal, most fit solutions and runtime. To come to these findings, we ran the number of items at various amounts of runs that scaled relative to the number of items tested.

The best bins for either GA or GA + WoC is a trivial measure, as with enough cycling, I would eventually get the more optimal response. With the genetic algorithm alone though, this seemed to be more common at first, until the higher item counts came into play. With the benefit of the extra runs that are native to the WoC, it was more common to find the most fit, optimal packing of bins within a single run.

On average**,** the Wisdom of Crowds approach performed many magnitudes slower than a single GA, however, less slightly less than the amount predicted by multiplying the standard unenhanced GA runtime by 10. When equipped on larger datasets, such as the 50-item one,

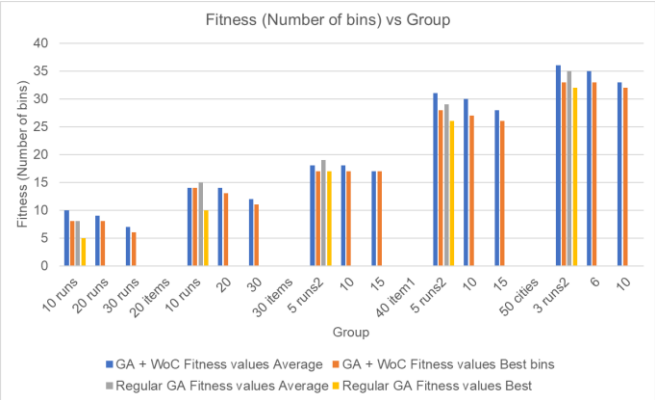
in most cases the WoC yielded a remarkably close to optimal solution, close to simply the GA. The more runs, the more efficient it was than a separate instance of the GA. Regardless, the WoC seemed to consistently perform slower however always ended up with a more optimal average with enough runs of the GA. As mentioned, the number of runs was dependent on the number of items, and so did the balance of runtime and optimization. Likewise, runtime correlated directly with the number of runs taken. As can be seen, the size of the problem made a difference in terms of range of optimization, when testing from 10 all the way to 50 items.

In some instances, the WoC would create an illegitimate solution, so the greedy fixer function would simply add bins to fix the overflow of displaced items.

Not unlike the last project and the implementation of TSP, the run times were a tad longer, so my comptuer did suffer another outburst of fans whirling.

It would be interesting to compare the performance of our evolutionary approach to another, non-GA, purely greedy approach. This would surely make an interesting comparison with our findings from the genetic approach we created here as well as the WoC. If the example set by TSP is anything to go by, then a greedy solution could operate at a degree faster of runtime than the GA or WoC, especially at item counts of well over 100.

The following are a bar chart representation of the fitness patterns as well as runtimes for each experimental group setting tested in Table 1:



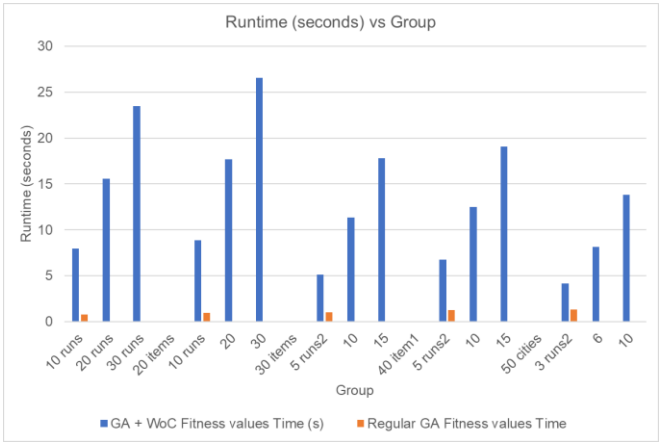


Fig. 3. (a) Fitness values displayed above. (b) Runtime values are displayed below.

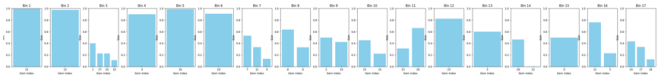


Fig. 4. GUI representation of bins filled; each bar is a single item, each plot a bin.

V. DISCUSSION

This time, we used a combination of our techniques and premade functions to help create the final program. In addition to our two-pronged approach for the GA, we used another three functions for the WoC feature: detecting segments of high overlap between the winners, creating a population with these apparent most fit segments hard coded, and then finally a clean-up function which ensures that the best route abides by the rules of bin completion, since it wouldn't be uncommon for repeated or skipped cities in some runs of WoC.

Compared to the greedy methods of our last project, the genetic algorithm was slower, as the process of a tournament is very lengthy and taxing. Another detail noticed was that there is sometimes an inconsistency with whether WoC will benefit, sometimes leading to bins with size greater than 1.00.

On the other hand, we found one of the best, most optimal algorithms for the bin sorting problem. The Bin Completion algorithm is a branch-and-bound strategy for one-dimensional, multi-container packing problems [18]. It starts with an upper bound, such as the best-fit decreasing solution, and applies a lower-bound heuristic function to prune the search. Rather than assigning items one at a time

to bins, it branches on the different maximal, feasible sets that can be assigned to each bin.

The Bin Completion algorithm combines a bin-oriented search space with a powerful dominance criterion that enables users to prune much of the space. This dominance criterion is based on the concept of undominated feasible sets. A set of elements is considered feasible if their sum does not exceed the bin capacity, and it is undominated if no other feasible set can pack all its elements into a set of bins whose capacities are the elements of the first set.

The performance of the basic bin completion framework can be enhanced by using a number of extensions, including no-good-based pruning techniques that allow further exploitation of the dominance criterion. On problems of size 60, bin completion runs more than a thousand times faster than another well-equipped, comparable algorithm: Martello and Toth's algorithm. In conclusion, the bin completion algorithm provides a powerful and efficient method for solving the bin packing problem, outperforming many existing algorithms in terms of speed and optimality.

In the future we hope we see an implementation for architecture-wise optimization such as multithreading, with each handling an individual tournament. While our final hybrid algorithm may not be as optimal or as efficient as the bin completion algorithm, we do know that this method is far superior to the search methods discussed in prior projects such as the first project's brute force methods. Because those were running at several seconds speed with only a dozen cities; at 30 and 40 cities, several milliseconds for the greedy algorithm though are nothing, which puts it at the top for quickest algorithms. The GA seems to scale linearly with the number of cities as well as generations ran, so a linear runtime is observed, not too dissimilar to the greedy. The number of tournaments run with their results compared for WoC is also direction related to the runtime. Some more areas for improvement would be an iteratively growing fitness graph with a GUI, as we could show how fitness decreased with each iteration; this would look pretty, as it would show the data live, as fitness improves. I believe that we could also use an optimization of break loop if the current tournament is looking like totaling greater than 1. We also would have liked to keep some other optimizing function at each step to allow the program to terminate early.

The interesting thing here is the way that WoC does not always lead to the most optimal outcome. You would think that WoC being another iteration of the GA search heuristic, would consistently allow that to occur, however this was not the case.

A GA and WoC approach can offer several benefits for solving the bin packing problem, which is a challenging NP-complete problem. First, a GA can explore a large and diverse search space of possible solutions, using crossover and mutation operators to generate new and improved solutions. Second, a WoC can aggregate the best solutions from multiple independent runs of the GA, using the most common bin assignments for each item as a way of exploiting the collective wisdom of the crowd. Third, a GA and WoC can be easily parallelized and distributed, as each run of the GA can be performed on a separate processor or machine, and the aggregation can be done by a simple voting scheme.

Despite the promising results of the GA and WoC approach, there are also some limitations and directions for future work. One limitation is that the GA and WoC may not always find the optimal or near-optimal solution, as they are heuristic methods that rely on randomness and approximation. Another limitation is that the GA and WoC may require many parameters, such as population size, mutation rate, crossover rate, number of generations, number of solutions, etc., which may affect the performance and robustness of the algorithm. Some possible future work includes comparing the GA and WoC with other existing algorithms for bin packing, such as the bin completion algorithm; applying the GA and WoC to other variants of the bin packing problem, such as the multidimensional or heterogeneous bin packing problem; and developing adaptive or self-tuning methods for setting the parameters of the GA and WoC.

The research conducted by researchers in the school of transportation and logistics engineering at the Wuhan University of Technology proposes a three-dimensional stacking model that considers both the utilization rate of pallet space and the stability criterion of goods as the objective function, and designs four constraints, including placement direction, pallet space and no overlapping. These can be used to formulate the bin packing problem as a multi-objective optimization problem with various constraints. It also proposes an improved genetic algorithm that uses the order of goods placement as the individual and designs a more reasonable crossover and mutation operator based on the order feature. This can be used to generate feasible and efficient solutions for the bin packing problem and improve the search performance and

convergence speed of the genetic algorithm. Comparing the results of the genetic algorithm with the traditional greedy algorithm and random order, the research shows that the genetic algorithm outperforms them in terms of fitness, number of pallets, and center of gravity ratio. This can be used to evaluate the effectiveness and robustness of the genetic algorithm for the bin packing problem and provide a benchmark for other algorithms [19].

The research also explores the impact of different weight coefficients, batch quantities, and size of goods on the results of the genetic algorithm and finds that a weight coefficient of about 0.1 is more suitable for the algorithm, and that the genetic algorithm is more stable and better than the greedy algorithm regardless of the quantity or size of goods. This can be used to guide the parameter setting and scenario selection of the genetic algorithm for the bin packing problem [19].

Further, research conducted at the Institute of Computer Science and Telecommunications at the Siberian State Aerospace University reviews recent advances in the domain of 2D irregular packing problems, which are a type of bin packing problem where the items have irregular shapes and orientations. The paper summarizes algorithms and strategies that have been proposed for the problems in recent years, such as heuristic algorithms, metaheuristic algorithms, hybrid algorithms, and exact algorithms. These algorithms can provide useful references and insights for solving the bin packing problem with Genetic Algorithm Approach and WoC. Their paper discussing the greedy heuristic for capacity planning details challenges and future directions of 2D irregular packing problems, such as improving the quality of solutions, reducing the computational time, handling dynamic and uncertain scenarios, and integrating with other optimization problems. These challenges can motivate further research and innovation for the bin packing problem with Genetic Algorithm Approach and WoC. It also provides a comprehensive bibliography of the literature on 2D irregular packing problems, which can be a valuable resource for researchers and practitioners who are interested in the bin packing problem with Genetic Algorithm Approach and WoC [20].

In this paper, we have presented a hybrid algorithm that combines a GA and a WoC approach to solve the challenging NP-complete problem, bin packing, which has many practical applications. Our algorithm uses a GA to generate and evolve solutions based on fitness, crossover, and mutation operators, and a WoC approach to aggregate the best solutions from multiple independent runs of the GA. We also use a greedy refinement function to ensure the validity and optimality of the final solution. We have tested our algorithm on various instances of the bin packing problem and compared its performance with a regular GA and a state-of-the-art bin completion algorithm. Our results show that our algorithm can find near-optimal solutions in reasonable time, and that the WoC approach can improve the quality of the solutions by exploiting the common patterns among the best solutions. We believe that our algorithm is a promising technique for solving the bin packing problem and other similar combinatorial optimization problems. We hope that our work can inspire further research and innovation in this field.

REFERENCES

[1] D. E. Goldberg, *Genetic Algorithms in search, optimization, and Machine Learning*. Reading, Mass.: Addison-Wesley Publishing Company, 1989

[2] K. Jansen, S. Kratsch, D. Marx, and I. Schlotter, "Bin packing with fixed number of bins revisited," *Journal of Computer and System Sciences*, vol. 79, no. 1, pp. 39–49, Feb. 2013, doi: https://doi.org/10.1016/j.jcss.2012.04.004.

[3] andretri, "Solving the Travelling Salesman Problem (TSP) Using Genetic Algorithms," *GitHub*, May 11, 2023. https://github.com/andretri/tsp-genetic-algorithm (accessed Nov. 27, 2023)

[4] J. J. Grefenstette, "How genetic algorithms work: A critical look at implicit parallelism," in Proceedings of the 3rd International Joint Conference on Genetic Algorithms (ICGA89), 1989.

[5] H. Aytug, M. Khouja, and F. E. Vergara, "Use of Genetic Algorithms to Solve Production and Operations Management problems: a Review," *International Journal of Production Research*, vol. 41, no. 17, pp. 3955–4009, Jan. 2003, doi: https://doi.org/10.1080/00207540310001626319.

[6] Richard E. Korf, "A New Algorithm for Optimal Bin Packing," in Proceedings of the Eighteenth National Conference on Artificial Intelligence, Edmonton, Alberta, Canada, 2002, pp. 731-736

[7] N. Mohamadi, "Application of Genetic Algorithm for the Bin Packing Problem with a New Representation Scheme," Mathematical Sciences, vol. 4, no. 3, pp. 253-266, 2010

[8] J. Surowiecki, The Wisdom of Crowds. New York: Knopf Doubleday Publishing Group, 2005

[9] M. Quiroz-Castellanos, L. Cruz-Reyes, J. Torres-Jimenez, C. Gómez S., H. J. F. Huacuja, and A. C. F. Alvim, "A Grouping Genetic Algorithm with Controlled Gene Transmission for the Bin Packing Problem," *Computers & Operations Research*, vol. 55, pp. 52–64, Mar. 2015, doi: https://doi.org/10.1016/j.cor.2014.10.010.

[10] Annu Malik, Anju Sharma, and Vinod Saroha, "Greedy Algorithm," International Journal of Scientific and Research Publications, vol. 3, no. 8, Aug. 2013

[11] G. Carmona-Arroyo, J. B. Vázquez-Aguirre and M. Quiroz-Castellanos, "One-Dimensional Bin Packing Problem: An Experimental Study of Instances Difficulty and Algorithms Performance," in Advances in Artificial Intelligence, Lecture Notes in Cocmputer Science, vol. 12565, pp. 123-136, Springer, 20211

[12] D. Stakic, A. Anokic and R. Jovanovic, "Comparison of Different Grasp Algorithms for The Heterogeneous Vector Bin Packing Problem," 2020 9th International Conference on Industrial Technology and Management (ICITM), Doha, Qatar, 2020, pp. 63-70, doi: 10.1109/ICITM48937.2020.9080298.

[13] D. S. Johnson, "Fast Algorithms for Bin Packing," *Journal of Computer and System Sciences*, vol. 8, no. 3, pp. 272–314, Jun. 1974, doi: https://doi.org/10.1016/s0022-0000(74)80026-7.

[14] S. K. M. Yi, M. Steyvers, M. D. Lee, and M. J. Dry, "Wisdom of the crowds in traveling salesman problems," J. Prob. Solv., vol. 3, no. 1, pp. 1–25, 2010

[15] S. P. Fekete, J. Grosse-Holz, P. Keldenich, and A. Schmidt, "Parallel Online Algorithms for the Bin Packing Problem," *Algorithmica*, vol. 85, no. 1, pp. 296–323, Sep. 2022, doi: https://doi.org/10.1007/s00453-022-01030-x.

[16] S. Albers, A. Khan, and L. Ladewig, "Best Fit Bin Packing with Random Order Revisited," *Algorithmica*, vol. 83, no. 9, pp. 2833–2858, Jul. 2021, doi: https://doi.org/10.1007/s00453-021-00844-5.

[17] C. Simoiu, C. Sumanth, A. Mysore, and S. Goel, "Studying the 'Wisdom of Crowds' at Scale," in Proceedings of the 18th AAAI Conference on Human Computation and Crowdsourcing (HCOMP), 2020, pp. 1-10.

[18] R. Baraglia, J. I. Hidalgo, and R. Perego, "A hybrid heuristic for the traveling salesman problem," IEEE Trans. Evol. Comput., vol. 5, no. 6, pp. 613-622, Dec. 2001

[19] S. Wu, S. Deng and J. Cao, "A Three-dimension Stacking Model with Modified Genetic Algorithm," 2022 IEEE International Symposium on Product Compliance Engineering - Asia (ISPCE-ASIA), 2022, pp. 979-987, doi: 10.1109/ISPCE-ASIA57917.2022.9971062.

[20] L. A. Kazakovtsev, M. N. Gudyma and A. N. Antamoshkin, “Genetic Algorithm with Greedy Heuristic for Capacity Planning,” 2014 6th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), St. Petersburg, 2014, pp. 607-613, doi: 10.1109/ICUMT.2014.7002150.