SOLVING GAP PROBLEM USING TABU SEARCH ALGORITHM IN PYTHON

IE 307
FALL 2024-2025

Dorukan Çatak, Gül Pembe Dağaşan, Onur Kurt, Şevval Naz Kırcı, Teoman Yıldırım

EXECUTIVE SUMMARY

The Generalized Assignment Problem (GAP) represents a significant optimisation challenge, necessitating the allocation of tasks to agents in a manner that minimizes costs while adhering to strict capacity constraints. This project investigates the use of the Tabu Search algorithm, a sophisticated metaheuristic approach that excels in exploring complex solution spaces and overcoming local optima (Glover, 1986). The Python-based implementation of Tabu Search developed for this project was tested on multiple GAP instances of varying sizes, ranging from small scenarios with 25 jobs and 5 agents to large ones with 50 jobs and 10 agents. The algorithm employs advanced techniques, including tabu list management, aspiration criteria, and neighborhood-based solution generation, to iteratively refine solutions. Parameters such as tabu tenure, maximum iterations, and stopping criteria were systematically tuned to optimise performance. Experimental results underscore the algorithm's efficacy in achieving substantial cost reductions. For instance, the Tabu Search algorithm achieved cost reductions compared to initial solutions. The execution times ranged from 0.0323 seconds for smaller instances to 0.5218 seconds for the largest instances, demonstrating its computational efficiency. Additionally, lower bounds were determined using linear programming (LP) relaxation as benchmarks, and the Tabu Search solutions consistently approached these bounds, highlighting the algorithm's effectiveness. This study highlights the versatility and scalability of the Tabu Search algorithm in addressing real-world resource allocation problems. Its ability to deliver high-quality solutions within acceptable computational times demonstrates its potential for broader applications in optimisation.

Key Words: Tabu search, Generalized Assignment Problem, Metaheuristic Algorithms, Resource Allocation, Cost Minimization, Capacity Constraints.

TABLE OF CONTENTS

.

1. INTRODUCTION

Optimization problems are crucial in decision-making processes across various domains, including logistics, manufacturing, and workforce management. Among these, the Generalized Assignment Problem (GAP) stands out as a complex yet practical NP-hard problem that requires assigning tasks to agents to minimize total costs while adhering to capacity constraints. Its applications span from scheduling and resource allocation to transportation planning, making it a cornerstone of operational research.

The GAP's complexity lies in balancing competing objectives: minimizing costs while satisfying capacity limitations. Due to their combinatorial nature, traditional optimization methods often struggle with large-scale GAP instances. To address this challenge, metaheuristic algorithms such as Tabu Search have emerged as effective solutions. Tabu Search extends local search methods by introducing memory-based mechanisms to explore new regions of the solution space and avoid cycling, thereby improving the chances of finding high-quality solutions (Glover, 1986).

In this project, a Python-based Tabu Search algorithm is developed and applied to solve GAP instances of varying sizes. The algorithm starts with an initial feasible solution and iteratively explores neighbouring solutions, guided by tabu restrictions and aspiration criteria. Key parameters such as tabu list length, stopping criteria, and neighbourhood structures are fine-tuned to balance computational efficiency and solution quality.

The project's primary objective is to evaluate the performance of Tabu Search in solving GAP and to identify the impact of algorithm parameters on solution quality. By conducting computational experiments across multiple problem instances, this study aims to provide actionable insights into the scalability, robustness, and practical applications of Tabu Search for real-world optimization challenges. The subsequent sections elaborate on the methodology, implementation, and results of the algorithm, highlighting its strengths and potential areas for further improvement.

2. LITERATURE REVIEW

The Generalized Assignment Problem (GAP) is a common combinatorial optimization problem that can be applied in numerous fields, such as resource allocation, supply chain management, and scheduling. GAP involves assigning jobs to m agents at a minimum cost while ensuring that each job is assigned to exactly one agent and resource constraints are not violated. To solve such a complex problem, we wanted to test heuristic and metaheuristic approaches such as Kernel Search, Simulated Annealing, and Tabu Search. In this section, a detailed review of these methodologies and their relevance to solving GAP is provided. The conclusion was made by justifying the choice of Tabu Search as the solution approach for the project.

2.1. Kernel Search

Kernel Search is a heuristic method widely used in solving large-scale optimization problems. This approach iteratively constructs and solves reduced subproblems to improve computational efficiency while preserving solution quality. In the context of GAP, (Boschetti & Vittorio Maniezzo, 2022) demonstrated that Kernel Search could find near-optimal solutions by dynamically identifying a subset of jobs, or "kernel," to be prioritized for assignment. This method's strength is its ability to balance exploration and exploitation. It focuses on computational effort on the most promising regions in the solution space.

On the other hand, Kernel Search requires the careful design of rules to define the kernel and manage its size. For GAP instances with numerous agents and jobs, these design choices can significantly affect performance. Scalability is the biggest problem of Kernel Search when dealing with highly constrained resource capacities. So, it is less appealing for specific problem instances.

2.2. Simulated Annealing

Simulated Annealing (SA) is a probabilistic metaheuristic inspired by the annealing process in metallurgy. The algorithm explores the solution space by allowing non-improving moves with a probability that decreases over time, controlled by a temperature parameter. When (Osman, 1995) applied Simulated Annealing to GAP, they observed and reported that the method works well on medium-sized examples, especially if it is hybridized with local search techniques.

The greatest strength of SA is its ability to escape local optima. This feature makes it suitable for complex optimization landscapes like GAP. On the other hand, its convergence to high-quality solutions depends especially on the cooling schedule and initial temperature settings. Also, SA requires a large number of iterations to achieve satisfactory results which means it has high computational costs for larger GAP examples. After considering these limitations and the requirement for extensive parameter tuning, SA seems less desirable for the project compared to other methods.

2.3. Tabu Search

Tabu Search (TS) is a metaheuristic designed to enhance local search by using memory structures to avoid revisiting the previously visited solutions. Glover (1989) explains TS as a flexible and robust approach to solving combinatorial optimization problems. For GAP, (Laguna et al., 1995) created a TS-based algorithm that incorporates adaptive memory and diversification strategies, demonstrating its effectiveness in finding high-quality solutions across various problem instances.

One of the key features of TS is the usage of a tabu list to store recently visited solutions or moves. It prevents cycling and helps the exploration of new regions in the solution space. Also, aspiration criteria allow the algorithm to override tabu restrictions if a move results in a significant improvement. TS is especially suitable to GAP because it can handle constraints through penalty functions and dynamic adjustment of search parameters.

Several studies have highlighted the scalability and adaptability of TS for solving large-scale Generalized Assignment Problems. Additionally, the method's straightforward implementation and minimal parameter requirements make it a practical choice for the GAP project. Considering all these characteristics and performance, we decided to use Tabu Search to solve our generalized assignment problem.

Aspect	Tabu Search (TS)	Kernel Search (KS)	Simulated Annealing (SA)
Strength	Avoids local optima, good solution quality.	Focuses on promising areas, efficient.	Escapes local optima effectively.
Weakness	Computationally slower for large problems.	Relies on good initial kernel selection.	Performance depends on cooling schedule.
Use Case	Resource allocation, scheduling problems.	GAP with well-defined solution clusters.	Problems needing exploration of broad areas.
Customization	Highly adaptable to problem specifics.	Moderate flexibility for GAP constraints.	Requires parameter tuning for efficiency.

Table 1. (Comparative Analysis of Each Method)

2.4. Comparative Analysis and Justification

The review of Kernel Search, Simulated Annealing, and Tabu Search highlights the strengths and weaknesses of each approach. Kernel Search excels in focusing computational effort on critical subsets of jobs but struggles with scalability for highly constrained instances. Simulated Annealing offers robust exploration capabilities but requires extensive parameter tuning and high computational costs. In contrast, Tabu Search provides a balanced trade-off between exploration and exploitation, with proven effectiveness in handling constraints and scalability.

Since we have three GAP instances of varying sizes and complexities, we require a heuristic that can adapt to different problem scales while maintaining solution quality. Tabu Search meets these criteria, offering a reliable and efficient framework for addressing GAP.

Therefore, Tabu Search is adopted as a final method, benefiting from its flexibility and performance advantages to develop a robust solution approach.

3. SOLUTION METHOD

Specifically in this section, the method to solve GAP which is the main subject of this thesis is explained in such a manner that the flow and steps of the algorithm are emphasized more than the programming details of the software. To address the identified problem, the deployed approach introduces heuristics to solve it optimally within the problem constraints and purpose. The flowchart visualizes the tabu search algorithm and illustrates the sequence of the algorithm, given in Appendix 1.

3.1 Problem: Problem Representation and Initial Solution

The GAP implies that jobs should be assigned to the agents in a way that the overall cost function will be least and none of the agents overloads its resource constraints. The nature of the problem is captured by a cost matrix, resource usage matrix, and the capacity vector of agents. The cost matrix yields the cost of a certain job with an agent where each cell has this job–agent cost combination, while the resource consumption matrix gives the resource usage of such an assignment. The capacity vector determines how many resources, at maximum, each agent can possess.

During the solution process, the first feasible solution is built, which is then further refined. Each job is taken one at a time, and for each job, the algorithm checks the resource utilization and availability of all the agents in an attempt to make the best allocation. The chosen agent is determined based on the process of minimizing the overload, which is calculated by the difference between the estimated resource utilization and the corresponding capacity of the agent. In case no feasible assignment is possible for a particular job, the algorithm assigns it to the maximum remaining capacity available in the agents so that all jobs are assigned.

3.2 Optimization Through Tabu Search

To optimize the search, the authors apply the Tabu Search heuristic, which is appropriate in GAP as it is a combinatorial problem. Tabu Search replaces the single current solution with another one through the iterative improvement of the current assignment and its neighborhood. In particular, neighboring solutions are generated by swapping one task between two agents based on whether this changes the solution's overall resource demands.

The algorithm starts by evaluating the cost of the initial feasible solution: the total assigning cost and constraint violation cost where applicable. It assesses all possible neighbors in each iteration and chooses the best among those neighbors for the next solution. It demonstrates that even if the neighbor is not better than the current solution, it is chosen to escape local optima. A tabu list is used to exclude many recently examined solutions that the algorithm looks for. This list temporarily disables some of those moves, for example, assigning the job back to the original agent.

To complement the exploration, the algorithm uses an aspiration criterion, where tabu moves occur if the new solution is better than the stored optimum. This balance of exploration and exploitation makes it easier for the algorithm to move around the solution space. The identified stopping conditions include a specific maximum number of iterations and the absence of any improvements during a given number of iterations.

3.3 Obtaining Lower Bound

To find the lower bound of these problem instances, we implemented Linear Programming (LP) relaxation. This process involves relaxing our integer constraints of the original problem, transforming it into a linear programming model where assignment variables are allowed to take continuous values. Using the PuLP library in Python, we formulated the LP model that minimizes the total cost of assigning jobs, while satisfying the capacity constraints of each agent. This relaxed model is then solved to optimality, returning a

lower bound for the original integer problem. The lower bound serves as a benchmark for our Tabu Search algorithm.

3.4 Decision-Making Process

The overall cost of a solution is the sum of its assignment cost and the penalties for any constraint infringements. Assignment costs can be directly generated from the cost matrix, while penalty costs are obtained by determining how an agent has exceeded their capacity. The former raises the desirability of feasible solutions, while the latter applies proportional repercussions on any violations to maintain a focus on high-quality candidates.

Neighbor solutions are produced systematically whereby every job is allocated to a different agent, and each neighbor solution found is assessed for its feasibility and the cost associated with it. The feasibility check guarantees that all the resource constraints are observed, while the cost evaluation defines potentially effective solutions. Hence, by approaching the problem in this manner, the algorithm gradually optimizes the current solution to near-optimally assign employees to companies.

3.5 Convergence and Visualization

The convergence rate of the proposed algorithm is traced using the cost function over iterations. This is depicted graphically using a cost history graph, with one axis representing iterations and the other representing the total cost. It demonstrates how the algorithm evolves towards optimal solutions and also showcases the algorithm’s capacity to move out of local optima. To show the overall effectiveness and capability of the method, results from multiple problem instances are compared and reported.

4. COMPUTATIONAL ANALYSIS

4.1. Experimental Results

For the Tabu Search Algorithm applied to the Generalized Assignment Problem (GAP), we performed several experiments to evaluate its computational efficiency and solution quality. In our study, we executed the Tabu Search Algorithm in different GAP instances, where the agent and jobs differentiate also the capacity constraints and their given assignment job assignment cost. Our Tabu Search algorithm also has its parameters such as tabu tenure which controls the number of moves that stay in the tabu list, the stopping condition parameter 'no_improvement_limit', and the second stopping condition 'max_iterations'. The no improvement limit parameter stops the algorithm if no improvement is made after the given iteration limit, and the second 'max_iterations' parameter ensures that the algorithm stops on the max iteration limit.

In Table 2, we execute our tabu search algorithm on multiple instances, and regarding results are given in the table. The table captures the initial solution, initial cost, the best solution, best cost, lower bound, iterations until termination, and execution times. In Table 2, the parameters are set to 10 for 'tabu_tenure', 1000 for 'max_iterations', and 50 for 'no_improvement_limit'.

PROBLEM INSTANCES	INITIAL SOLUTION	INITIAL COST	BEST SOLUTION	BEST COST	LOWER BOUND	ITERATIONS UNTIL TERMINATION	EXECUTION TIME (S)
Problem Instance 1	[2, 3, 0, 2, 1, 4, 2, 0, 4, 3, 0, 1, 4, 3, 0, 0, 1, 2, 1, 4, 3, 3, 2, 0, 4]	519	[2, 3, 0, 0, 4, 1, 2, 3, 4, 3, 0, 1, 1, 1, 4, 0, 1, 2, 2, 3, 3, 0, 2, 4, 1]	450	431.8344	68	0.0322
Problem Instance 2	[3, 1, 4, 2, 4, 6, 5, 1, 0, 6, 7, 4, 2, 0, 7, 5, 2, 3, 4, 1, 7, 0, 5, 3, 2, 6, 3, 0, 2, 4, 5, 6, 6, 7, 3, 4, 1, 0, 2, 6]	790	[3, 4, 2, 2, 0, 7, 5, 3, 1, 6, 6, 4, 2, 0, 7, 0, 7, 3, 5, 1, 4, 3, 5, 2, 3, 6, 3, 0, 7, 1, 1, 6, 6, 0, 6, 7, 0, 6, 2, 6]	673	639.1367	79	0.1657
Problem Instance 3	[1, 4, 2, 0, 3, 1, 5, 8, 2, 7, 4, 3, 9, 2, 0, 4, 6, 5, 2, 1, 6, 9, 3, 0, 7, 8, 1, 5, 2, 1, 6, 5, 7, 6, 7, 8, 0, 9, 3, 4, 9, 0, 5, 4, 6, 8, 4, 1, 2, 7]	845	[1, 3, 3, 4, 5, 3, 5, 9, 2, 7, 4, 4, 2, 4, 4, 2, 1, 9, 1, 1, 2, 7, 2, 0, 0, 4, 1, 1, 0, 6, 4, 5, 9, 6, 8, 2, 7, 3, 0, 8, 8, 9, 5, 9, 6, 3, 6, 9, 5, 7]	594	567.5132	118	0.5218

Table 2. (Output of Tabu Search Algorithm)

In Table 3, we adjust our tabu tenure parameter to 8 and run the same problem instances again to investigate the performance of the algorithm. All of the other parameters are set exactly the same with the first run. In Table 3, the parameters are set to 8 for ‘tabu_tenure’, 1000 for ‘max_iterations’, and 50 for ‘no_improvement_limit’.

PROBLEM INSTANCES	INITIAL SOLUTION	INITIAL COST	BEST SOLUTION	BEST COST	LOWER BOUND	ITERATIONS UNTIL TERMINATION	EXECUTION TIME (S)
Problem Instance 1	[2, 3, 0, 2, 1, 4, 2, 0, 4, 3, 0, 1, 4, 3, 0, 0, 1, 2, 1, 4, 3, 3, 2, 0, 4]	519	[2, 3, 0, 0, 4, 1, 2, 3, 4, 3, 0, 1, 1, 1, 4, 0, 1, 2, 2, 3, 3, 0, 2, 4, 1]	450	431.8344	68	0.0323
Problem Instance 2	[3, 1, 4, 2, 4, 6, 5, 1, 0, 6, 7, 4, 2, 0, 7, 5, 2, 3, 4, 1, 7, 0, 5, 3, 2, 6, 3, 0, 2, 4, 5, 6, 6, 7, 3, 4, 1, 0, 2, 6]	790	[3, 4, 2, 2, 0, 7, 5, 3, 1, 6, 6, 4, 2, 0, 7, 0, 7, 3, 5, 1, 4, 3, 5, 2, 3, 6, 3, 0, 7, 1, 1, 6, 6, 0, 6, 7, 0, 6, 2, 6]	673	639.1367	79	0.1671
Problem Instance 3	[1, 4, 2, 0, 3, 1, 5, 8, 2, 7, 4, 3, 9, 2, 0, 4, 6, 5, 2, 1, 6, 9, 3, 0, 7, 8, 1, 5, 2, 1, 6, 5, 7, 6, 7, 8, 0, 9, 3, 4, 9, 0, 5, 4, 6, 8, 4, 1, 2, 7]	845	[1, 3, 2, 4, 5, 3, 5, 8, 2, 7, 4, 4, 8, 0, 6, 2, 1, 9, 1, 1, 2, 7, 2, 0, 0, 6, 1, 1, 0, 8, 4, 5, 9, 6, 8, 2, 7, 3, 3, 5, 9, 9, 5, 9, 6, 3, 6, 9, 4, 7]	606	567.5132	84	0.3906

Table 3. (Output of Tabu Search Algorithm)

4.2. Interpretation of the Results

The computation starts with the initial solution generated by our pre-defined method, assigning jobs to employees with the most remaining capacity. Tabu Search Algorithm continues to search the neighbors, and if it finds a better cost updates the solution. In Figure 1, we plot the results of Table 2. The results show that when the problem size increases the computation time increases. Given that our problem instances are relatively small, our tabu search algorithm is executed in a relatively short time.

Referring to Table 2, our initial cost for problem instance 1 is 519, the best cost is 450, and its lower bound found with LP relaxation equivalent to 431.83. The difference between

the lower bound and the best cost is relatively small, and the iteration count to find the best solution is 68 with 0.0322 seconds of runtime.

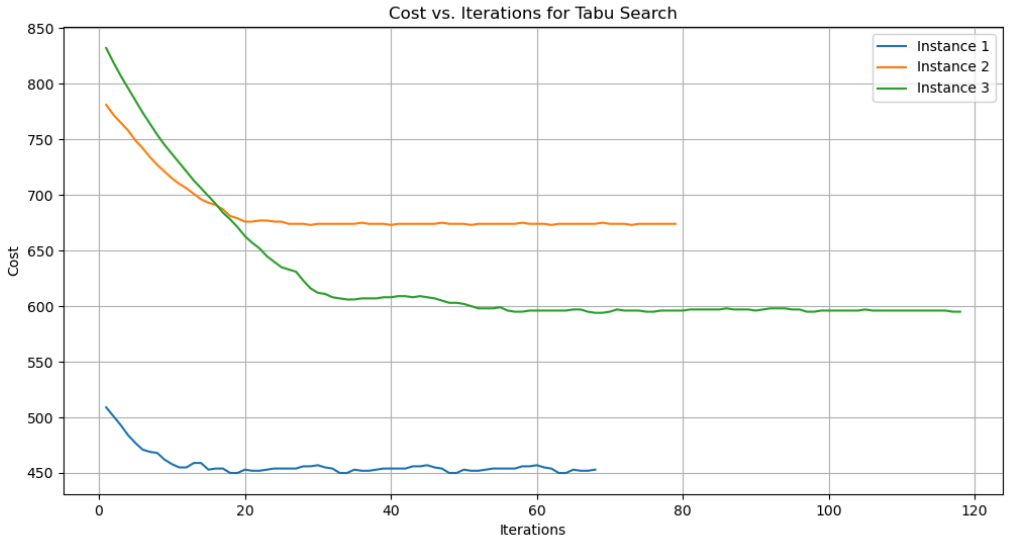


Figure 1. (Costs vs. Iterations for Tabu Search)

On the second problem instance, our problem size increases slightly; we have 8 agents with 40 jobs to assign. The initial solution for problem instance 2 has a cost of 790 and our best cost found with the tabu search algorithm is 673. The lower bound for problem instance 2 is 639.13. Furthermore, the iteration count for the second problem instance is 79 and the computation time is 0.1657 seconds.

For the third problem instance, our problem size increases to 10 agents with 50 jobs to assign. The initial solution's cost for the third problem instance is 845. The best cost is 594, and the lower bound is 567.51. The iteration count is 118, and the computation time is 0.5218 seconds.

The results in Figure 1. show that our tabu search algorithm finds the best costs close enough to the lower bound obtained with LP relaxation. Furthermore, the iteration count and computation times show that when the problem size increases, the compute time and iteration count to obtain the best cost also increase.

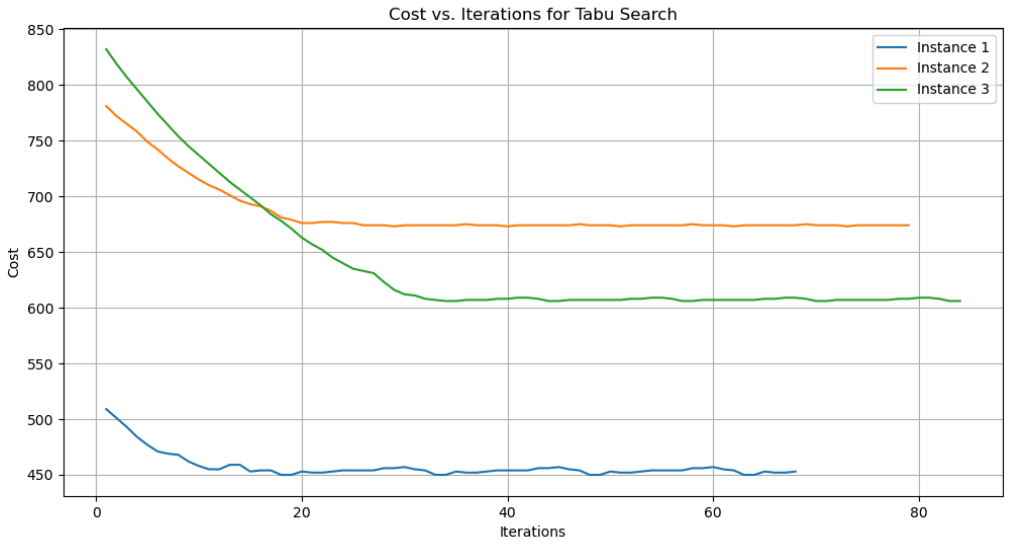
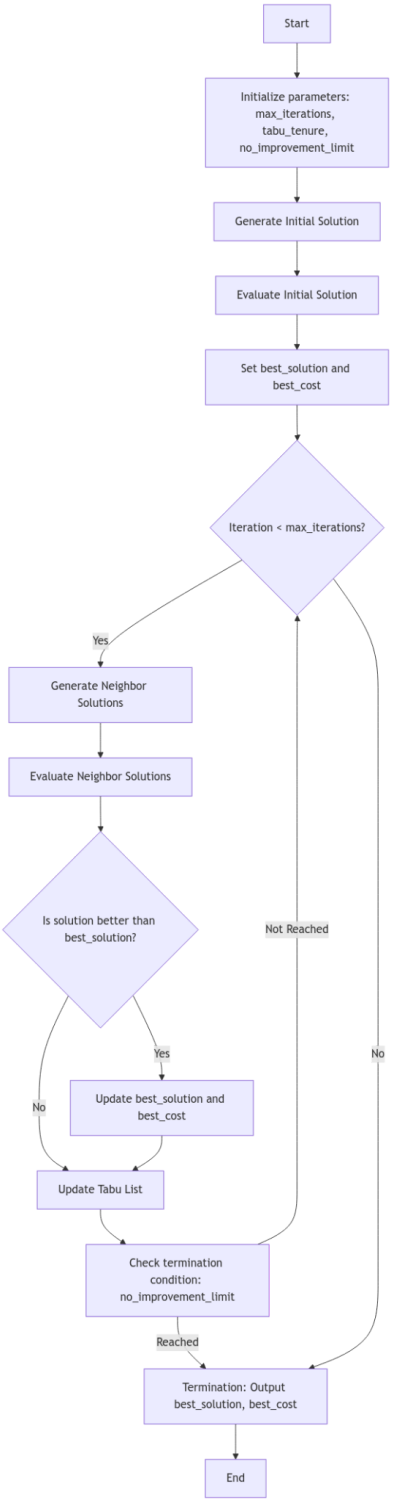


Figure 2. (Costs vs. Iterations for Tabu Search)

In Figure 2, the Tabu Tenure parameter is set to 8, and the Tabu Search algorithm is executed again. The results from Table 3, show that the problem instance 1 and 2 have not been affected by that change. However, the third problem instance tabu search algorithm found the best cost at the 84th iteration with a 0.3906-second runtime, the iteration count on the first execution for problem instance 3 is 118 with a 0.5218-second runtime. The execution time and the iteration count were reduced, on the other hand, the best cost obtained increased to 606.

APPENDIX 1: Flow Chart of Tabu Search Algorithm



APPENDIX 2

In the first week, our group of five members focused on gaining a thorough understanding of the Generalized Assignment Problem (GAP). We collectively researched the literature to comprehend the problem and its applications, accompanied by examples. This process also helped us gradually decide on the methodology to use, eventually leading to the selection of the Tabu Search algorithm. During this phase, Dorukan Çatak organized a shared Drive folder to establish a structured workflow, ensuring a smooth progression for the project.

In the second week, Onur Kurt initiated the coding phase by beginning the implementation of the algorithm. Dorukan Çatak contributed by refining and editing the code. All group members engaged in discussions to understand and improve the algorithm's development. Concurrently, Gül Pembe Dağaşan examined solutions from similar projects to support the enhancement of the code.

By the third week, challenges associated with the Tabu Search method arose, leading to some uncertainties. In response, Şevval Naz Kırcı revisited and updated the literature review section to address these concerns. After reviewing the updated insights, the group reaffirmed its decision to proceed with Tabu Search.

In the fourth week, Teoman Yıldırım initiated the preparation of the project presentation. He divided the presentation into sections and assigned each group member their respective parts to prepare. After completing the presentation, the group collectively analyzed the entire project to finalize the content for the progress presentation.

During the fifth week, feedback from the progress presentation was incorporated into the project. Under the leadership of Dorukan Çatak, the entire group focused on improving the code and addressing identified gaps.

In the sixth week, the report writing phase began under the leadership of Şevval Naz Kırcı and Gül Pembe Dağaşan. The report sections were distributed among the group members, with each individual responsible for drafting their assigned parts.

Finally, in the seventh week, Onur Kurt and Teoman Yıldırım led the efforts to review the code and the report, ensuring both were in their final forms. The group members collaborated by asking each other questions to refine the work further, ultimately preparing for the project demo.

This structured approach allowed for efficient collaboration, ensuring that all aspects of the project were thoroughly addressed and completed on time.

REFERENCES

Boschetti, M. A., & Vittorio Maniezzo. (2022). Matheuristics: using mathematics for heuristic design. 4OR, 20(2), 173–208. https://doi.org/10.1007/s10288-022-00510-8

DíazJ. A., & Fernández, E. (2001). A Tabu search heuristic for the generalized assignment problem. European Journal of Operational Research, 132(1), 22–38. https://doi.org/10.1016/s0377-2217(00)00108-9

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. Computers & Operations Research, 13(5), 533–549. https://doi.org/10.1016/0305-0548(86)90048-1

Glover, F. (1989). Tabu Search—Part I. ORSA Journal on Computing, 1(3), 190–206. https://doi.org/10.1287/ijoc.1.3.190

Laguna, M., Kelly, J. P., JoséLuis González-Velarde, & Glover, F. (1995). Tabu searches for the multilevel generalized assignment problem. European Journal of Operational Research, 82(1), 176–189. https://doi.org/10.1016/0377-2217(93)e0174-v

Osman, I. H. (1995). Heuristics for the generalized assignment problem: simulated annealing and tabu search approaches. OR Spectrum, 17(4), 211–225. https://doi.org/10.1007/bf01720977