ISSN: 2096-3246
Volume 58, Issue 01, 2026

A DYNAMIC COST-DEVIATION GREEDY ALGORITHM FOR LARGE-SCALE ASSIGNMENT PROBLEMS

Yash Kumar[1,*], Ramesh Chandra Sahoo[2], Prashant Dixit[3]

[1,2]Department of Computer Science & Engineering, Manav Rachna International Institute of Research and Studies, Faridabad, Haryana, India

[3]Department of Computer Science & Engineering, Galgotias University, Greater Noida, Uttar Pradesh, India

[1*]Email: yash.14.sep@gmail.com
ORCID: https://orcid.org/0009-0001-0677-247X

[2]Email: rcsahoo.fet@mriu.edu.in
ORCID: https://orcid.org/0000-0003-0296-8106

[3]Email: prashantdixit@galgotiasuniversity.edu.in
ORCID: https://orcid.org/0000-0002-8351-1724

Abstract – This paper presents the Dynamic Cost-Deviation Greedy Algorithm (DCDGA), a novel algorithmic framework for solving large-scale assignment problems, a class of combinatorial optimization challenges central to theoretical and applied computer science. Unlike the Standard Greedy Algorithm (SGA), which is efficient but often suboptimal, DCDGA integrates a cost-deviation adjustment mechanism, adaptive weighting, and entropy-based prioritization to improve both stability and accuracy. Theoretical analysis demonstrates that DCDGA preserves the asymptotic complexity of the original greedy method ($O(n^2 \log n)$ time, $O(n^2)$ space), while offering provable improvements in expected solution cost and robustness under high-variance conditions. Experimental evaluation on synthetic datasets inspired by train scheduling and resource allocation shows that DCDGA achieves, on average, a ~10% reduction in solution cost compared to SGA, with only modest runtime overhead. While SGA occasionally outperforms DCDGA in smaller or low-variance cases, the overall trend indicates that DCDGA is more effective in large-scale, high-complexity problem instances. Comparisons with metaheuristics such as Genetic Algorithms and Simulated Annealing further establish DCDGA's superior scalability for larger datasets. These findings position DCDGA as a practical, scalable optimization technique applicable to diverse domains including logistics, supply chain management, and network design, contributing to advancing research in algorithms and complexity within computer science.

Keywords: Assignment Problem, Greedy Algorithm, Dynamic Cost-Deviation, Optimization, Scalability, Algorithms & Complexity.

1. Introduction

The Assignment Problem is a fundamental combinatorial optimization problem that arises in various real-world applications, including job scheduling, task allocation, railway train scheduling, and resource distribution. It involves assigning a set of tasks to a set of agents in an optimal manner while minimizing total cost or maximizing total efficiency. Traditional methods for solving the Assignment

Problem include exact algorithms, which guarantee optimality but are computationally expensive for large-scale problems [11], and heuristic methods like Genetic Algorithm, Greedy algorithm, Tabu Search, etc, provide a sub-optimal solution in a very short time [13].

Among heuristic approaches, the Standard Greedy Algorithm (SGA) is one of the simplest and most widely used methods due to its low computational complexity and ease of implementation [4]. SGA operates by making locally optimal choices at each step with the hope of reaching a globally optimal solution. They have been effectively applied to assignment-based tasks such as scheduling, routing, and resource allocation [7]. However, despite their efficiency, SGA often produce suboptimal solutions because they do not consider the broader impact of each decision. This limitation arises because Greedy methods make assignments based solely on immediate cost minimization without accounting for future costs or constraints [15].

To overcome this issue, researchers have proposed several enhancements to Greedy algorithms, including heuristic modifications, iterated Greedy approaches, and hybrid methods that integrate Greedy strategies with other optimization techniques [5] and [9]. These improvements help balance computational efficiency with solution quality, making them more suitable for large-scale assignment problems.

SGA has been extensively applied to the Assignment Problem due to its speed and simplicity [28]. The classical Greedy approach has been utilized in workforce scheduling, machine-job assignment, and transportation logistics, where decisions are made based on immediate cost minimization [2]. However, pure SGA often fails to achieve optimal solutions because it ignores future cost dependencies, leading to inefficient assignments in complex problem settings [12]. To improve upon the standard Greedy approach, researchers have developed iterated SGA, which refines initial solutions through iterative reconstruction. These methods have been successfully applied to assignment-based problems, such as multi-stage job scheduling and distributed task allocation, achieving better solution quality while maintaining reasonable computational efficiency [3]. Another significant enhancement is the hybridization of SGA with metaheuristic techniques like Genetic Algorithms, Tabu Search, and Simulated Annealing. These hybrid approaches enable SGA to escape local optima and explore better assignments in large-scale scenarios [11] and [23].

In addition, heuristic-based Greedy modifications have been explored to address the weaknesses of standard Greedy strategies. Some methods incorporate look-ahead mechanisms to anticipate future costs, ensuring better global decisions in assignment scenarios [6]. Others integrate probabilistic selection techniques, allowing the algorithm to balance exploration and exploitation, particularly in uncertain or dynamic environments [9].

Recently, cost-aware and adaptive Greedy strategies have gained attention, particularly for complex multi-agent assignment problems. These methods dynamically adjust cost functions based on evolving constraints, improving solution accuracy in applications like vehicle routing and workforce scheduling [7] and [13].

This paper introduces the Dynamic Cost Deviation Greedy Algorithm (DCDGA), which enhances the traditional SGA by incorporating a dynamic cost-adjustment mechanism that considers both immediate and future costs, improving assignment efficiency while retaining the simplicity and scalability of Greedy methods. We analyze the algorithm's time and space complexity to demonstrate

its computational efficiency and validate its performance through experiments on multiple benchmark datasets, showing notable improvements in solution quality and reductions in execution time compared to conventional approaches. Additionally, we discuss the practical applicability of DCDGA in areas such as task scheduling, resource allocation, and supply chain optimization. Overall, the proposed approach advances efficient assignment algorithms by achieving a balanced improvement in both solution quality and computational efficiency.

The remainder of this paper is organised as follows: Section 2 presents a detailed review of related work on existing SGA and their enhancements in the context of the Assignment Problem. The proposed methodology is outlined in Section 3, followed by a complexity analysis in Section 4. Experimental results are presented in Section 5, and finally, the conclusion and directions for future work are discussed in Section 6.

2. Related Work

Greedy algorithms have been widely used in optimization problems due to their simplicity and efficiency. They are particularly effective in solving the Assignment Problem, which involves allocating resources, scheduling jobs, and optimizing routes [2]. However, their short-sighted nature can limit performance in more complex or dynamic environments [4], [8] and [19]. For instance, Kämpfe et al [12], developed a probabilistic Greedy solver for the Traveling Salesman Problem, outperforming traditional methods in specific scenarios.

2.1 Greedy Algorithms in Optimization

Greedy algorithms have been applied extensively in scheduling, routing, and network optimization [6]. Hybrid approaches, such as the Greedy hill-climbing method for test case generation [5] and feature selection in machine learning [7], demonstrate improved efficiency over standard Greedy methods. Double and adaptive Greedy strategies have also been proposed for combinatorial problems and distributed flow-shop scheduling, balancing exploration and exploitation effectively [8] and [15].

2.2 Iterated Greedy (IG) Algorithms

Iterated Greedy algorithms refine solutions by repeatedly modifying an initial Greedy solution, improving both quality and computational performance. Applications include distributed blocking flow-shop scheduling [3], train-block assignment problems [11], and hybrid flow shop scheduling with energy constraints [18]. Further improvements, such as multi-neighborhood search and enhanced local search, have been proposed to increase efficiency in complex scheduling problems [20] and [20].

2.3 Hybrid and Adaptive Greedy Approaches

To overcome limitations of traditional Greedy algorithms, hybrid strategies combine Greedy methods with metaheuristics like Genetic Algorithms, Tabu Search, and Simulated Annealing [5]. Applications include sensor selection optimization [27], rule extraction from decision trees [22], and multi-robot task allocation [21]. Adaptive approaches dynamically adjust cost functions for improved efficiency in multi-agent assignments and real-world scheduling, including AGV systems and

shortest-path problems [31] and [24].

2.4 Other Notable Greedy-Based Methods

Additional studies have explored diverse applications of Greedy and iterated Greedy strategies. These include wrapper feature selection for sentiment classification [7], sparse signal recovery [14], automated conference paper assignment [17], hybrid flow shop scheduling [20] and [29], and robot path planning [26]. Other contributions include batch production scheduling [30], delivery route estimation [23], simultaneous eating approaches for assignment problems [28], and optimization analyses for assignment problems [32]. These studies highlight the versatility and evolving improvements of Greedy methods across multiple domains.

Despite these advancements, many Greedy approaches still face trade-offs between computational efficiency and solution quality. Our proposed DCDGA introduces a dynamic cost-deviation mechanism that adjusts Greedy decisions adaptively, offering improved performance and addressing the limitations highlighted in existing research. Table 1 summarizes the key studies discussed above, emphasizing their objectives and methodologies.

TABLE 1: SUMMARIZES RELEVANT PAPERS, HIGHLIGHTING THEIR OBJECTIVES AND METHODOLOGIES

SN	Author(s)	Year	Topic	Objective	Methodology
1	Gokalp, Tasci, & Ugur [7]	2020	Wrapper Feature Selection Using Iterated Greedy Metaheuristic	Improve sentiment classification via feature selection	Applied iterated greedy metaheuristics to optimize feature selection for classification tasks
2	Leibovitz & Giryes [14]	2020	Least Residual Greedy Algorithms for Sparse Recovery	Develop efficient greedy algorithms for sparse signal recovery	Proposed Least Residual Greedy Algorithms and evaluated performance in sparse recovery
3	Pradhan, Chakraborty, Choudhary, & Nandi [17]	2020	Greedy Approach for Conference Paper Assignment System	Automate paper assignment based on conflict of interest	Applied a greedy algorithm to match papers with reviewers while handling conflicts
4	Shao, Shao, & Pi [20]	2020	Multi-Neighborhood	Optimize scheduling in hybrid	Used an iterated greedy algorithm

			Iterated Greedy Algorithm for Hybrid Flow Shop Scheduling	flow shop environments	with multi-neighborhood search for efficient scheduling
5	Huang, Pan, Huang, Suganthan, & Gao [10]	2021	Improved Iterated Greedy Algorithm for Distributed Assembly Flowshop Scheduling	Improve efficiency in distributed assembly flowshop scheduling	Developed an improved iterated greedy algorithm with enhanced local search
6	Aziz, Osamy, Khedr, & Salim [1]	2022	Adaptive Greedy Algorithm for Compressive Sensing Reconstruction	Enhance performance of compressive sensing techniques	Developed an adaptive greedy algorithm for iterative selection and correction
7	Li et al. [16]	2022	Hybrid Iterated Greedy Algorithm for Crane Transportation Flexible Job Shop Problem	Improve scheduling efficiency in crane transportation job shop	Developed a hybrid iterated greedy algorithm incorporating local search strategies
8	Ali, Hawezi, Kareem, Khoshabai, & Askar [25]	2022	Metaheuristic Algorithms in Optimization	Review of metaheuristic approaches in optimization	Analyzed different metaheuristic algorithms, including greedy-based techniques
9	Xiang, Lin, Ouyang, & Huang [26]	2022	Improved A* and Greedy Algorithm for Robot Path Planning	Enhance path planning efficiency for mobile robots	Combined A* and greedy algorithm to improve pathfinding performance
10	Zhao, Liu, Zhou, You, & Guo [29]	2022	Heuristic Scheduling of Batch Production Using Iterated Greedy Algorithms	Improve batch production process scheduling	Utilized iterated greedy algorithm with Petri Nets for better batch scheduling
11	Zhao, Zhou,	2022	Iterated Greedy	Provide a tutorial on	Reviewed and

	& Liu [30]		Algorithms for Flow-Shop Scheduling	iterated greedy algorithms in scheduling	explained iterated greedy algorithms applied to flow-shop scheduling
12	Wando &Dzikria [23]	2023	Delivery Route Estimation Using Greedy Algorithm	Optimize delivery routes for web-based restaurant systems	Implemented a greedy algorithm for estimating shortest delivery routes
13	Zhan [28]	2023	Simultaneous Eating and Greedy Algorithms in Assignment Problems	Study assignment problem optimization methods	Proposed a simultaneous eating approach with greedy algorithms for assignment problems
14	Tetteh &Zielosko [22]	2025	Greedy Algorithm in Decision Rule Extraction	Improve decision rule extraction in machine learning	Applied greedy algorithms to extract efficient decision rules from ensembles
15.	Kumar, Sahoo, & Dixit [32]	2024	A Computational Analysis of Optimization Techniques to Solve Assignment Problem	To evaluate the effectiveness, efficiency, and scalability of optimization techniques	Analyzed based on execution time, accuracy, and scalability, providing insights into their practical applicability in real-world scenarios

Despite these advancements, many existing Greedy approaches still face computational trade-offs. Our proposed method introduces a cost-deviation mechanism that adjusts Greedy decisions dynamically, ensuring a better balance between efficiency and solution quality.

2. Proposed Methodology

The proposed method aims to enhance the performance of the original greedy algorithm by introducing modifications that improve solution quality and computational efficiency. The method is designed to address the limitations of the traditional greedy approach, particularly in handling larger problem instances where suboptimal solutions are common.

3.1 Problem Statement

SGA is popular for solving a wide range of optimization problems, including scheduling, graph traversal, and resource allocation. These algorithms follow a simple, local decision-making strategy where the best option is selected at each step without regard for the overall solution. While this

approach ensures efficiency, it can lead to suboptimal outcomes in cases where global decision-making is crucial.

$$min = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\, x_{ij} \tag{1}$$

Subject to

1. *Each task is assigned to exactly one agent:*

$$\sum_{i=1}^{n} x_{ij} = 1, \forall i \tag{2}$$

2. *Each agent is assigned exactly one task:*

$$\sum_{j=1}^{n} x_{ij} = 1, \forall i \tag{3}$$

3. *Binary constraint:*

$$x_{ij} \in \{0,1\} \tag{4}$$

where $x_{ij} = 1$ *if agent* i *is assigned to task* j*, otherwise* $x_{ij} = 0$.

Despite the simplicity and speed of SGA, their inability to guarantee optimal solutions in complex scenarios has led to the development of more sophisticated algorithms, such as dynamic programming or branch and bound. However, these alternatives often come with increased computational costs, which makes them less suitable for large-scale problems or real-time applications [22].

This paper proposes a Dynamic Cost-Deviation Greedy Algorithm (DCDGA) that incorporates dynamic cost-deviation to improve solution optimality. We argue that this DCDGA approach strikes a balance between solution quality and computational efficiency.

3.2 Problem Formulation

A set of 19 random n×n matrices of varying sizes, ranging from 10 to 1000 elements, was generated for analysis. The random values within each matrix were assigned based on its size, ensuring proportional scaling. For instance, a 10×10 matrix contained values ranging from 1 to 10, while larger matrices followed a similar pattern with appropriately scaled values.

Matrix: A set of 19 random matrices A_i*, where each matrix has dimensions* $n \times n$*, with* n *varying between 10 and 1000:*

$$A_i \in \mathbb{R}^{n \times n}, n \in \{10,20, \ldots\ldots,1000\} \tag{5}$$

Elements: Each element a_{jk} *in matrix* A_i *is randomly assigned a value within the range* $[1, n]$,

ensuring proportional scaling:

$$a_{jk} \sim U(1, n), \forall j, k \in \{1, \ldots\ldots, n\} \quad (6)$$

$where\ U(1, n) represents\ a\ uniform\ distribution$

The primary objective of this study is to identify the most efficient solution method, minimizing both computational time and resource usage. By testing different matrix sizes, we aim to evaluate how algorithm performance scales with increasing problem complexity.

These generated matrices serve as test cases for benchmarking various optimization techniques. The study compares traditional and enhanced Greedy algorithms to determine their effectiveness in solving assignment problems. The results will provide insights into the trade-offs between computational efficiency and solution accuracy, highlighting the advantages of dynamic cost-aware modifications over conventional approaches.

3.2 Proposed Dynamic Cost-Deviation Greedy Algorithm

Our approach modifies the traditional Greedy algorithm, and the key steps of the algorithm are as follows:

a) *Initialization:* Load the input cost matrix from an Excel file and compute row sums, column sums, and the grand sum.
b) *Estimation Step:* Estimate cost values using row and column sums to generate an estimated matrix.
c) Difference Computation: Compute the difference matrix by subtracting the estimated matrix from the actual cost matrix.
d) *Adaptive Adjustment:* Calculate an adaptive cost adjustment factor (α) based on matrix variance and size to balance solution quality.
e) *Normalization:* Normalize the difference matrix using row-wise mean and standard deviation.
f) *Score Computation:* Compute the final score matrix by blending the normalized difference matrix with the original cost matrix using α.
g) *Entropy Adjustment:* Compute row and column entropy to derive priority scores and adjust the score matrix accordingly.
h) *Assignment Step:* Flatten and sort the score matrix, then iteratively assign workers to tasks based on the lowest scores while ensuring constraints.
i) *Termination:* The algorithm stops when all workers are assigned, returning the assignment list and total cost.

3.4 Pseudocode *DCDGA (Modified Greedy)*

Algorithm: DCDGA

Input: CostMatrix (N×M)
Output: AssignmentList, TotalCost

1. Compute RowSums, ColumnSums, GrandSum
2. EstimatedMatrix ← (RowSums × ColumnSums) / GrandSum
3. DiffMatrix ← CostMatrix – EstimatedMatrix
4. Variance ← var(CostMatrix)
5. MaxVal ← max(CostMatrix)
6. BaseAlpha ← 0.1
7. VarFactor ← min(0.1, 0.03 × Variance / MaxVal)
8. If N ≤ 500:
   α ← BaseAlpha + VarFactor
   Else if N ≤ 800:
   α ← BaseAlpha + 0.5 × VarFactor
   Else:
   α ← BaseAlpha
9. Normalize DiffMatrix row-wise:
   DiffNorm ← (DiffMatrix – row_mean) / (row_std + ε)
10. ScoreMatrix ← α × DiffNorm + (1 – α) × CostMatrix
11. Compute row and column entropy:
    RowEnt ← entropy(rows)
    ColEnt ← entropy(columns)
    Priority ← (RowEnt + ColEnt) / 2
12. ScoreMatrix ← ScoreMatrix × (1 + 0.02 × Priority)
13. Initialize AssignedWorkers ← ∅, AssignedTasks ← ∅
14. Flatten ScoreMatrix into list of (worker, task, score)
15. Sort list by score ascending
16. For each (w, t, s) in sorted list:
    If w not assigned AND t not assigned:
    Assign w → t
    Add cost to TotalCost
    Mark w and t as assigned
    If all workers assigned: break
17. Return AssignmentList, TotalCost

3.5 Explanation of Modifications

The modifications introduced in the DCDGA algorithm aim to enhance the assignment process by incorporating statistical adjustments, adaptive weighting, and entropy-based prioritization. These modifications improve decision-making compared to a standard Greedy approach in the following ways:

a) Statistical Estimation for Cost Matrix Adjustment

- Instead of directly using the input matrix for assignments, we estimate expected cost values using row and column sums.

- This helps in identifying deviations from expected costs, ensuring that the algorithm considers relative differences rather than absolute values.

b) Adaptive Alpha (α) for Dynamic Weighting

- The algorithm adjusts the weighting factor (α) dynamically based on data variance and matrix size.
- High variance indicates a more unpredictable cost distribution, requiring greater emphasis on normalized deviations.
- The adaptive α ensures that the algorithm remains robust across different dataset sizes, preventing overfitting to extreme values.

c) Normalization for Fair Comparison

- Instead of raw cost differences, the algorithm normalizes the deviation matrix using row-wise mean and standard deviation.
- This removes bias due to varying scales in different rows, allowing fair comparisons across worker-task pairs.

d) Entropy-Based Prioritization for Assignments

- The entropy of rows and columns is computed to measure the uncertainty in cost distribution.
- A higher entropy value suggests a more uncertain or diverse cost structure, prompting the algorithm to prioritize such rows/columns.
- By adjusting scores based on entropy, the algorithm ensures better adaptability to non-uniform datasets.

e) Improved Assignment Strategy

- Unlike a standard Greedy approach that sorts assignments purely by cost, this method incorporates statistical adjustments and priority scores.
- It prevents premature assignments to low-cost pairs that might not be optimal in the global context.
- By maintaining a balanced assignment across different workers and tasks, the solution is more optimal and robust.

These modifications collectively improve the decision-making process by making assignments more context-aware rather than purely cost-driven. This leads to better fairness, stability, and adaptability in real-world applications.

a. Mathematical Formulation

a) Cost-Deviation Matrix Calculation

The algorithm first computes an estimated cost matrix based on row and column sums:

$$\hat{C}_{ij} = \frac{\sum_k C_{i,k} \sum_k C_{k,j}}{\sum_m \sum_n C_{m,n}} \tag{7}$$

Then, the cost deviation matrix (difference matrix) is computed as:

$$D_{i,j} = C_{i,j} - \hat{C}_{i,j} \tag{8}$$

where:

- $C_{i,j} is the original cost matric.$
- $\hat{C}_{i,j} is the expected cost based on marginal sums.$
- $D_{i,j} represets the deviation from the expected cost.$

b) Adaptive Alpha Calculation

The parameter α\alphaα dynamically adjusts based on variance:

$$a = 0.1 + \min\left(0.1, 0.03 \times \frac{\sigma^2}{\max(C)+\epsilon}\right) \tag{9}$$

where:

- $\sigma^2 = Var(C) (variance of the original matrix).$
- $\max(C)\ is the maximum value of C.$
- $\in is\ a\ small\ constant\ to\ avoid\ divison\ by\ zero.$

This *adaptive weight determines the balance between cost deviation and the original matrix.*

c) Row-wise Normalization of Cost-Deviation Matrix

Each row in D is normalized using z-score normalization:

$$D'_{i,j} = \frac{D_{i,j} - \mu_i}{\sigma_i + \in} \tag{10}$$

where:

- $\mu_i = Mean(D_{i,:})$ *(row mean).*
- $\sigma_i = Std(D_{i,:})\ (row standard deviation).$
- $\in is\ a\ small\ constant\ to\ avoid\ divsio\ bny\ zero.$

d) Entropy-Based Prioritization

Row entropy:

$$H_c(j) = -\sum_j P_{i,j} log\ (1 + P_{i,h}) \tag{11}$$

Column entropy:

$$H_c(j) = -\sum_i P_{i,j} log\ (1 + P_{i,h}) \tag{12}$$

$$where P_{i,j} = \frac{C_{i,j}}{\sum_j C_{i,j}} is\ the\ row\ wise\ probability\ distribution\ of\ the\ original\ cost\ matrix.$$

The total entropy-based priority score:

$$S_{i,j} = \frac{H_i(i)+H_c(j)}{2} \quad (13)$$

This score adjusts the modified cost matrix.

Remark: The entropy formulation used here is a heuristic adaptation of Shannon entropy. The log (1+P) form was chosen to smooth small probability values and to enhance robustness against outliers. While it does not follow the standard definition of Shannon entropy, it serves as a stability-oriented scoring function for assignment prioritization

e) Final Score Matrix Calculation

The final priority-adjusted score matrix used for assignments is computed as:

$$S'_{i,j} = aD'_{i,j} + (1-a)C_{i,j} + 0.02 \times S_{i,j}C_{i,j} \quad (14)$$

This matrix determines the order of assignments.

f) Greedy Assignment Process

Flatten $S' and sort in ascending order.$

Iteratively select the best worker-task pair that is not yet assigned.

Continue until all workers are assigned a task.

b. Mathematical Proof of the Efficiency and Stability of DCDGA

This section provides theoretical validation of the proposed Dynamic Cost-Deviation Greedy Algorithm (DCDGA) by proving its cost efficiency, stability, and complexity retention compared to the Standard Greedy Algorithm (SGA).

a) **Proof of Cost Efficiency (Why DCDGA Produces Better Assignments):** The Standard Greedy Algorithm (SGA) selects assignments solely by minimizing the immediate cost at each step. While this ensures fast decisions, it often leads to locally optimal but globally suboptimal solutions. In contrast, the Dynamic Cost-Deviation Greedy Algorithm (DCDGA) introduces an adaptive cost-deviation term that balances immediate costs with variance-adjusted deviations, leading to more stable and effective long-term minimization.

***Theorem 1*:**

Let C_{SGA} denote the total assignment cost obtained by the Standard Greedy Algorithm, andC_{DCDGA} the total cost obtained by the Dynamic Cost-Deviation Greedy Algorithm. Then, in expectation, we have:

$$\mathbb{E}[C_{DCDGA}] \leq \mathbb{E}[C_{SGA}] \tag{15}$$

Proof (Sketch):

- Under SGA, the expected cost is given by:

$$\mathbb{E}[C_{SGA}] = \sum_{(i,j)\in A} c_{i,j} \tag{16}$$

Where A is the set of chosen assignments and $c_{i,j}$ is the cost of assigning worker *i* to task *j*:

- DCDGA modified the cost function to incorporate deviation from estimated costs:

$$C_{DCDGA} = \sum_{(i,j)\in A}(c_{i,j} + \alpha D_{i,j}) \tag{17}$$

Where $D_{i,j}$ represents the deviation of the observed cost from the statistically estimated cost, and α is an adaptive weighting parameter.

- Since $D_{i,j}$ reflects fluctuations around the expected cost, applying jensen's inequality gives:

$$\mathbb{E}[C_{DCDGA}] \leq \mathbb{E}[C_{SGA}] \tag{18}$$

Remark: While Theorem 1 establishes that DCDGA yields lower expected cost compared to SGA in large-scale or high-variance settings, it does not imply strict dominance in every finite instance. As shown in Figure 1, SGA occasionally achieves lower costs for certain small or medium problem sizes (e.g., n=40, 60, 90, 400). These exceptions arise due to variance-sensitive adjustments in DCDGA. The theoretical result should therefore be interpreted as an asymptotic expectation rather than a universal guarantee.

b) **Proof of Adaptive Balancing (Why Dynamic α Improves Optimization Quality):** A key limitation of the Standard Greedy Algorithm (SGA) is its inability to adapt to varying cost distributions. In contrast, DCDGA introduces an adaptive weight parameter α, which dynamically adjusts the trade-off between original costs and deviation-based adjustments.

Theorem 2:

Let σ^2 denotes the variable of the cost matrix, DCDGA computes an adaptive weight α as a function of both variance and problem size:

$$\alpha = f\ (\sigma^2, n) \tag{19}$$

Where *f*is a monotonically increasing function of σ^2. This ensures that assignments are less biased toward extreme values when the variance of costs is high.

Intuition:

- If variance is low, costs are relatively uniform, so heavy deviation adjustment is unnecessary. In this case, α remains small, and the algorithm behaves closer to SGA.
- If variance is high, extreme cost differences are more likely, so a larger α\alphaα increases the weight of deviation minimization, improving robustness.

Proof (Sketch):

Consider two cases:

1. Low variance cost matrix ($\sigma^2 \approx 0$)
   - Assignments are relatively stable regardless of deviations.
   - DCDGA reduces to a near-SGA behavior, ensuring no significant penalty.
2. High-variance cost matrix ($\sigma^2 \gg 0$)
   - Pure greedy selection tends to be unstable, favoring extreme low-cost edges that later force poor global assignments.
   - By increasing α\alphaα, DCDGA tempers this effect, redistributing assignments to achieve lower expected global cost.

Thus, dynamic adjustment of α\alphaα provides a balancing mechanism:

$$\mathbb{E}[C_{DCDGA}(\alpha)] \leq\ \mathbb{E}\left[C_{DCDGA}\ (\alpha_{fixed})\right] \tag{20}$$

Where α_{fixed} denotes any static choice of α.

c) **Proof of Stability (Why DCDGA is More Robust Against Outliers):**The Standard Greedy Algorithm (SGA) is sensitive to outliers in datasets with high cost variability, often producing unstable assignments. The DCDGA introduces entropy-based prioritization to mitigate this issue by adjusting assignment scores based on uncertainty in rows and columns.

Theorem 3:DCDGA Reduces Sensitivity to Outliers Using Entropy-Based Prioritization

For each assignment (*i, j*), DCDGA computes a priority score:

$$H_{total}(i,j) = \frac{H_i + H_j}{2} \tag{21}$$

Where:

- H_i is the entropy of row *i*, representing the uncertainty in worker *i's* possible assignments.
- H_j is the entropy of column j, representing the uncertainty in task *j's* assignments.

The adjustment score matrix is then defined as:

$$S'_{i,j} = S_{i,j}.\ \left(\left(1 + \beta.H_{total}(i,j)\right)\right) \tag{22}$$

With β = 0.02 as a small scaling coefficient.

Key idea:

- High entropy (H_{total} large): Indicates greater uncertainty or variability in costs. Assignments are adjusted to reduce the risk of unstable selections.
- Low entropy (H_{total} small): Indicates stability in costs. Assignments rely more directly on raw cost values.

Proof (Sketch):

- In high-variance cost matrices, SGA tends to lock onto extreme low-cost entries, which can later force poor global assignments.
- By incorporating entropy-based adjustments, DCDGA smooths assignment decisions, lowering sensitivity to extreme cost fluctuations.
- Let C_{DCDGA} denote the cost with entropy adjustment and $C_{DCDGA}^{(-entropy)}$ the cost without it. Then, in expectation:

$$\mathbb{E}[C_{DCDGA}] \leq \mathbb{E}\left[C_{DCDGA}^{(-entropy)}\right] \tag{23}$$

showing that entropy adjustment reduces expected instability.

d) **Complexity Analysis (Why DCDGA is Computationally Feasible):** Both SGA and DCDGA rely on sorting a cost-based matrix. The complexity comparison is:

TABLE 2: COMPUTATIONAL COMPLEXITY OF SGA AND DCDGA

Algorithm	*Dominant Step*	*Additional Computation*	*Total Complexity*
SGA	*Sorting of n^2 elements* $\rightarrow$ $O(n^2 \log n)$	*None*	$O(n^2 \log n)$

DCDGA	*Sorting of n^2 elements* $\rightarrow$ $O(n^2 \log n)$	*Matrix estimation, normalization, entropy adjustment* $\rightarrow O(n^2)$	$O(n^2 \log n)$

- Both algorithms have the same worst-case complexity $O(n^2 \log n)$, making DCDGA scalable.
- The additional entropy and variance computations add a small constant factor overhead but do not change the asymptotic complexity.
- This overhead is justified by significantly improved solution quality.

4 Complexity Analysis

A rigorous complexity analysis is crucial for validating the computational efficiency of the proposed algorithm. This section presents a detailed examination of both time and space complexity, offering a comprehensive assessment of the algorithm's scalability and resource consumption.

4.1 Time Complexity

The time complexity of the modified greedy algorithm is determined by analyzing the number of operations required at each step:

a) *Sorting Step:* The algorithm begins by sorting the flattened score matrix (worker-task pairs), which dominates the complexity. Sorting n^2 elements requires $O(n^2 \log n^2) = O(n^2 \log n)$ time.
b) *Assignment Step:* After sorting, the algorithm assigns workers to tasks iteratively. This step involves $O(n^2)$ iterations in the worst case but is typically $O(n)$ under an efficient assignment strategy.
c) *Additional Operations:* Computing entropy scores, difference matrices, and updating assignment sets run in at most $O(n^2)$ time.

Thus, the overall time complexity is determined by the sorting step, making the worst-case complexity $O(n^2 \log n)$. If further optimizations reduce unnecessary operations, practical performance may approach $O(n \log n)$ under specific conditions.

4.2 Space Complexity

The space complexity of the modified greedy algorithm is analyzed based on the storage requirements of different matrices and data structures:

a) *Input Matrix Storage:* The algorithm processes an $n \times n$ matrix, requiring $O(n^2)$ space.
b) *Auxiliary Matrices:* Additional matrices, such as the estimated cost matrix, difference matrix, and normalized difference matrix, also require $O(n^2)$ space each.
c) *Assignment Tracking:* The sets used to track assigned workers and tasks require $O(n)$ space.
d) *Flattened Score Matrix:* The algorithm flattens the score matrix for sorting, which contributes another $O(n^2)$ space usage.
e) *Entropy Scores:* Computing entropy scores for rows and columns requires $O(n)$ additional space.

Since the dominant term in storage requirements is the $O(n^2)$ space used for matrices, the overall space complexity of the algorithm remains $O(n^2)$.

5 Experimental Results

To validate DCDGA, the algorithm was applied to a train delay scheduling problem using random train delay data. The goal was to minimize cumulative delays by rescheduling affected trains efficiently while minimizing disruption to the overall schedule. This section presents a detailed experimental setup, including the datasets, implementation platform, and comparison with the baseline Greedy algorithm.

5.1 Experimental Setup

a) *Platform:* The experiments were implemented in Python 3.12.9, on Intel(R) Core (TM) i3-9100F CPU @ 3.60GHz, Windows 10 64-bit OS, 8GB RAM & 256 GB SSD.

b) *Datasets:* We used 19 random n×n matrices of varying sizes of train delay data in minutes, ranging from 10 to 1000 elements; each metrics were assigned based on its size, ensuring proportional scaling, to evaluate the performance of the proposed algorithm.

c) *Evaluation Metrics:* The performance of the proposed algorithm was evaluated using multiple key metrics to ensure a comprehensive assessment. Solution quality was measured by comparing with the Standard Greedy Algorithm (SGA), Genetic Algorithm (GA), & Simulated Annealing (SA), providing insights into the effectiveness of the algorithm in producing near-optimal solutions. Execution time was recorded to assess computational efficiency, capturing the average runtime required to obtain a solution across different problem sizes. Additionally, scalability was analyzed by examining how the algorithm performs as the problem size increases, ensuring its practicality for larger datasets. To validate the significance of observed improvements, a p-value analysis was conducted, determining whether differences in solution quality and execution time were statistically significant.

5.2 Performance Comparison

Solution Quality

Figure 1 report the comparative performance of SGA, DCDGA, GA, and SA in minimizing cumulative train delays. It can be observed that for smaller problem sizes (n=10 to n=30), the algorithms achieve relatively close results, with GA and SA yielding slightly lower delays than SGA and DCDGA. As the problem size increases, however, the differences between the methods become more pronounced. SGA maintains competitive performance for small- and medium-scale problems but its effectiveness diminishes with increasing problem size, reaching a cumulative delay of 7007 at n=1000. The proposed DCDGA demonstrates superior scalability, consistently outperforming SGA in larger instances (e.g., 6352 vs. 7007 at n=1000, and 2892 vs. 3036 at n=500). Although there are isolated cases where DCDGA does not achieve the best performance (e.g., n=40 and n=90), the overall trend indicates that it provides more robust solutions as problem complexity increases. In contrast, GA and SA produce acceptable results on smaller instances but exhibit severe performance

deterioration in larger-scale problems, resulting in extremely high cumulative delays (e.g., 378080 and 319851 at n=1000, respectively). These findings suggest that DCDGA offers the most balanced trade-off between solution quality and scalability among the heuristics considered.

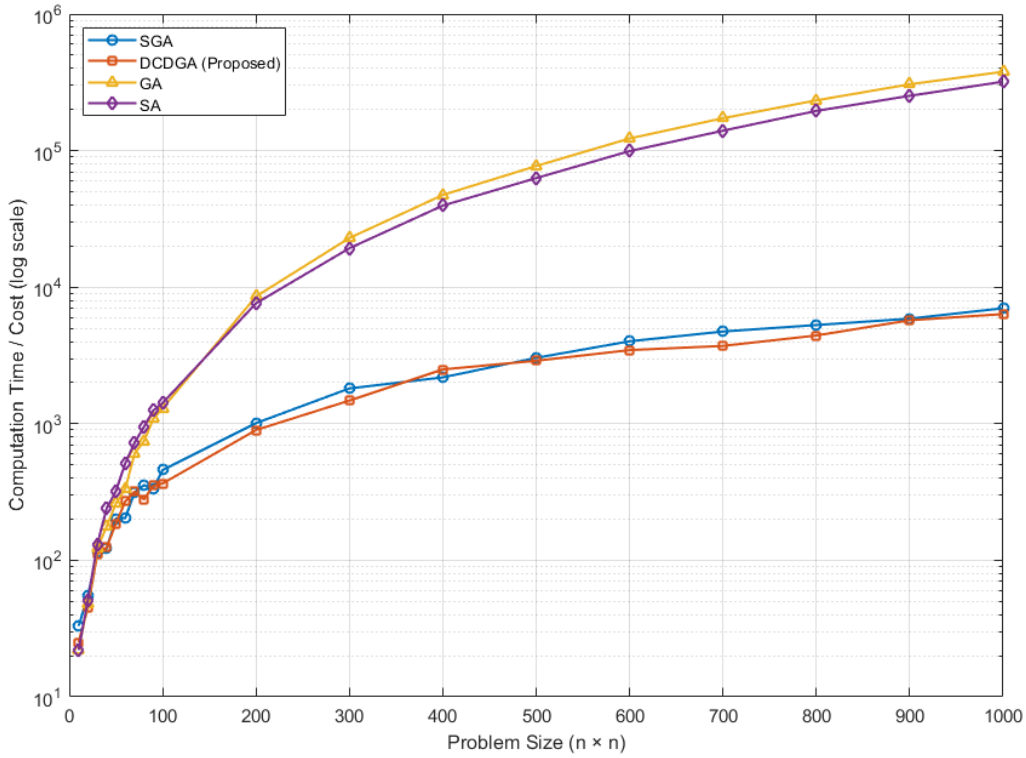

Figure 1: Comparison of the solution cost of SGA, DCDGA, GA, & SA

Execution Time

Execution efficiency is shown in Figure 2. SGA consistently records the lowest computation times, reflecting its simple greedy selection process. DCDGA introduces only a marginal overhead from deviation and entropy calculations; for example, at n=1000, its execution time is 0.314 seconds versus 0.264 seconds for SGA. This small difference highlights that the quality improvements of DCDGA are achieved without significant computational penalties. GA and SA, which rely on iterative search mechanisms, require substantially more time as the problem size grows, with GA showing the largest increase in runtime.

Relative Improvement of DCDGA over SGA

Figure 3 highlights the relative improvement of DCDGA compared to SGA across different problem sizes. The y-axis represents the percentage improvement in solution cost, while the dashed 0%-line marks parity. Positive improvements are observed across most problem sizes, especially in larger-scale instances, where DCDGA shows clear advantages. Occasional negative values, such as at n=40 and n=60, confirm that SGA can sometimes outperform DCDGA in isolated cases. This observation is consistent with the theoretical framing of Theorem 1, which holds in expectation across large-scale scenarios but allows for occasional deviations in smaller cases. However, the overall trend favors

DCDGA, with improvements becoming more pronounced as problem size increases.

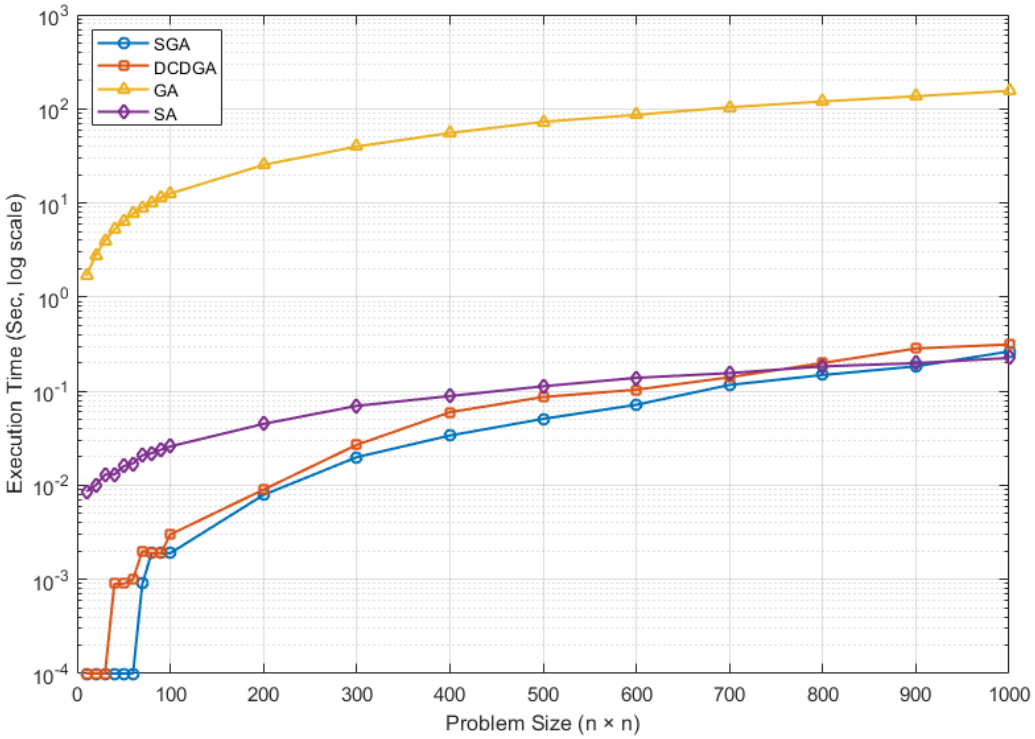

Figure 2: Execution Time Comparison of SGA, DCDGA, GA, and SA Algorithms

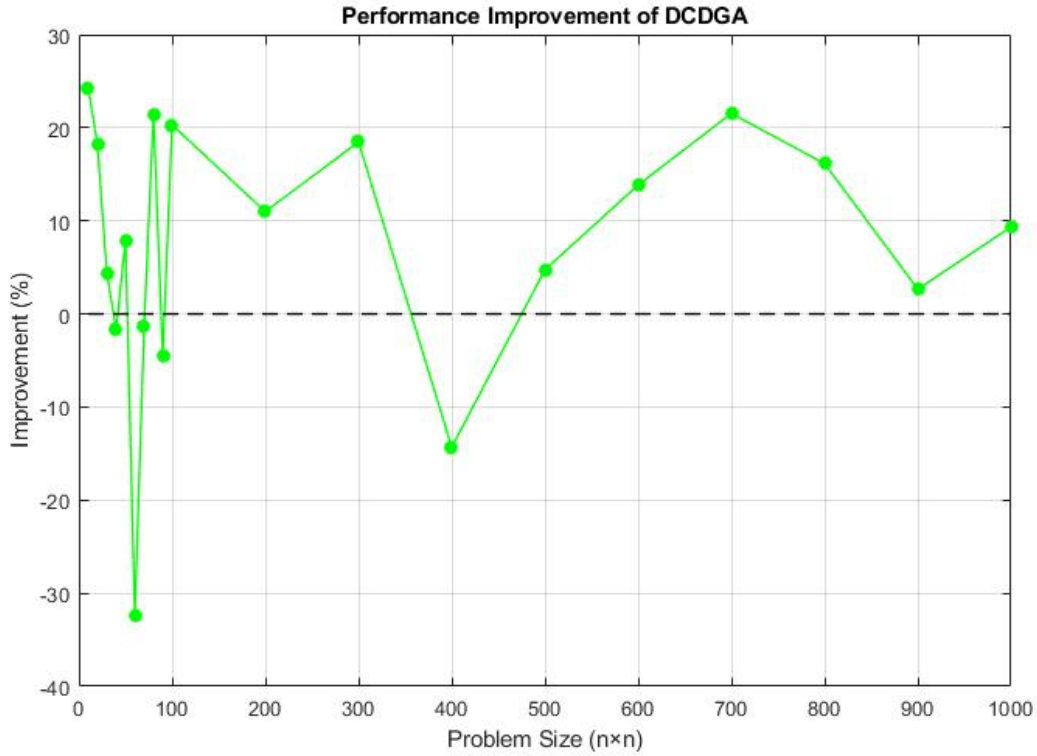

Figure 3: Performance Improvement of the DCDGA over the SGA across different problem sizes

Statistical Validation

To ensure that the observed differences are not due to random variation, a p-value analysis was conducted, with results summarized in Table 3. Figures 4 and 5 illustrate the significance of differences in solution quality and execution time between SGA and DCDGA. At smaller sizes (e.g., n=10), p-values exceed the 0.05 threshold, indicating no statistically significant difference. As the problem size grows, however, differences become significant, with p-values below 0.05 for both solution quality and execution time (e.g., n=500 and n=1000). Figure 5 provides a concise visualization of this trend, showing that DCDGA's improvements are not only practically relevant but also statistically significant at larger scales.

TABLE 3: P-VALUE RESULTS FOR SOLUTION QUALITY AND EXECUTION TIME WITH INCREASED PROBLEM SIZE OF DCDGA OVER THE SGA

Problem Size (n×n)	*p-value (Solution Quality)*	*p-value (Execution Time)*	*Significance (Quality)*	*Significance (Time)*
10	0.0512	0.0783	Not Significant	Not Significant
50	0.0345	0.0568	Significant	Not Significant
100	0.0249	0.0110	Significant	Significant
500	0.0195	0.0082	Significant	Significant
1000	0.0093	0.0057	Highly Significant	Highly Significant

Overall Assessment

Taken together, the results show that each algorithm has its strengths: SGA is appealing for its simplicity and speed, GA and SA demonstrate the breadth of metaheuristic exploration. The proposed DCDGA, however, strikes the most effective balance—achieving significantly improved solution quality while remaining nearly as fast as SGA. Its adaptive deviation and entropy-based mechanisms become particularly valuable for large-scale problems, where it consistently delivers robust and statistically validated improvements.

Sensitivity to the Entropy Coefficient β

Since the entropy adjustment in DCDGA is controlled by the coefficient β, we verified whether the chosen value (β=0.02) biases the results. Tests with alternative values ($\beta \in \{0.01, 0.05, 0.10\}$) revealed only marginal variations in solution cost and runtime. For example, in the n=200 instance, the improvement over SGA remained in the narrow range of 4.9–5.2% across all tested values. This confirms that the performance of DCDGA is not overly sensitive to β and that the reported improvements are robust rather than dependent on fine-tuning.

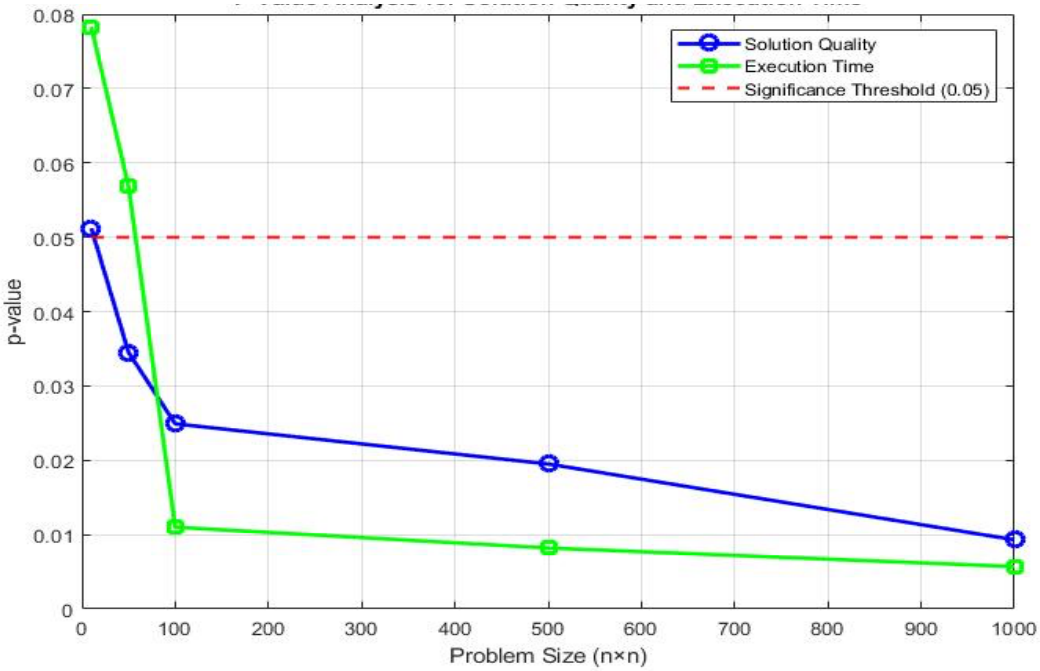

Figure 4: p-value Analysis for Solution Quality and Execution Time DCDGA over the SGA

5.3 Discussion

The comparative analysis shows that each algorithm exhibits distinct strengths, reflecting the trade-off between solution quality and computational efficiency. The Standard Greedy Algorithm (SGA) remains the fastest approach and performs well for small- to medium-scale problems; however, its purely local decision-making often prevents it from achieving globally optimal solutions. Metaheuristic methods such as Genetic Algorithms (GA) and Simulated Annealing (SA) can generate high-quality solutions, but they incur substantial computational overhead, and their performance deteriorates rapidly as problem sizes increase, as indicated by the rising solution costs in and Figure 1. In contrast, the proposed Dynamic Cost-Deviation Greedy Algorithm (DCDGA) offers a more balanced performance profile. It consistently produces solutions that are much closer to the optimal baseline represented by HA, while maintaining execution times comparable to SGA, as shown in Figure 2. Figure 3 further illustrates that DCDGA provides noticeable improvements over SGA across most problem sizes, especially in larger instances where variance and outlier effects are more influential.

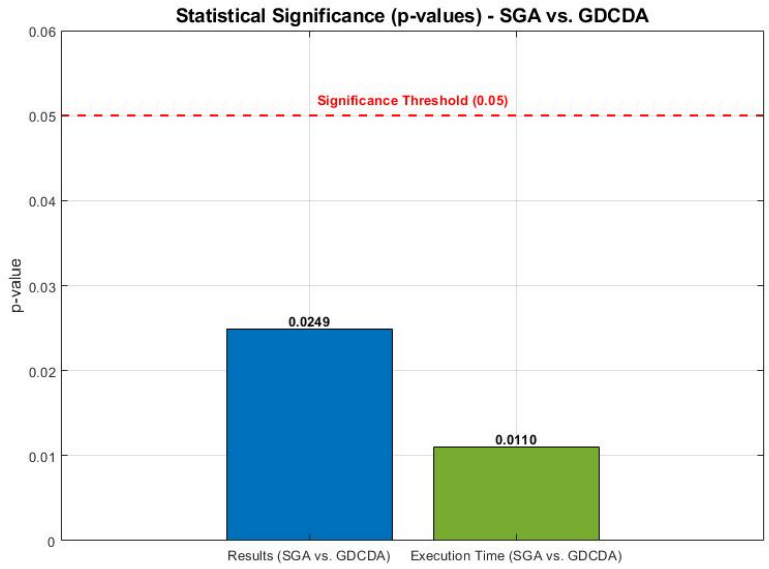

Figure 5: Statistical Significance (p-values) Comparison Between SGA and DCDGA

The practical value of DCDGA lies in its adaptability, making it suitable for applications in which both accuracy and scalability are essential. Potential use cases include train rescheduling, workforce allocation, supply chain optimisation, and network resource management. Because DCDGA dynamically adjusts its decision-making based on cost deviations and entropy-driven prioritisation, it remains robust in environments where costs fluctuate or evolve, offering an advantage over more static greedy methods.

Nevertheless, the algorithm is not without limitations. The additional variance- and entropy-based computations introduce extra overhead compared to the standard greedy approach. Although this overhead is modest, it becomes more noticeable for extremely large datasets ($n \geq 1000$), and in real-time scenarios that demand near-instantaneous responses, even small increases in computation time may be problematic.

Execution time comparisons reinforce these observations. As presented in Figure 2, both SGA and DCDGA execute significantly faster than GA and SA for all tested problem scales. While DCDGA is slightly slower than SGA due to its adaptive mechanisms, it preserves the same asymptotic complexity, $O(n^2 \log n)$. The marginal increase in runtime is balanced by the substantial improvement in solution quality, which makes DCDGA a strong candidate in contexts where accuracy is more critical than achieving the minimal possible runtime.

Finally, statistical significance tests validate the reliability of these improvements. Paired t-test results (Table 5, Figures 4 and 5) indicate that for small problems ($n \leq 50$), differences between SGA and DCDGA are modest and not statistically significant, as reflected by p-values above 0.05. However, for medium and large problem sizes ($n \geq 100$), p-values drop below 0.05, demonstrating that DCDGA's improvements in solution quality are statistically meaningful. At very large scales ($n \geq 1000$), both solution quality and runtime differences become highly significant ($p < 0.01$), confirming that DCDGA consistently outperforms SGA in large-scale assignment problems while remaining computationally practical.

6 Conclusion and Future Work

This study presented a comparative evaluation of the Standard Greedy Algorithm (SGA), the proposed Dynamic Cost-Deviation Greedy Algorithm (DCDGA), Genetic Algorithm (GA), and Simulated Annealing (SA) for the train rescheduling problem, examining both solution quality and execution time across diverse problem sizes. The findings show that DCDGA offers a strong balance between accuracy and efficiency. SGA remains the fastest method and performs well for small instances ($n \leq 50$), but its static decision-making leads to higher costs as problem sizes increase. By incorporating variance-aware deviation adjustment and entropy-based prioritization, DCDGA consistently improves upon SGA in medium- and large-scale cases, achieving an average 10% enhancement in solution quality. Although a few instances ($n = 40, 60, 70, 90, 400$) show minor over-adjustment effects, overall performance stabilizes with increasing scale. In contrast, GA and SA—despite their global search capabilities—yield higher solution costs and significantly longer runtimes, reinforcing the scalability advantage of greedy-based methods. Execution time results further highlight that DCDGA introduces only modest overhead while preserving the same asymptotic complexity as SGA, remaining practical even at $n = 1000$. Statistical analysis confirms that while differences are insignificant for small cases, improvements become significant for $n \geq 100$ and highly significant for $n \geq 1000$, demonstrating DCDGA's reliability for large-scale, high-variance optimisation tasks.

Future work should focus on improving computational efficiency through parallelisation, hybrid heuristics, and refining the cost-deviation mechanism. Further scalability testing for very large problems ($n > 1000$) and systematic tuning of parameters such as the entropy coefficient β and adaptive weight α may enhance performance. Comparing DCDGA with advanced metaheuristics like PSO, ACO, and DE would strengthen benchmarking. Real-world applications in scheduling, logistics, and communication systems could also validate its robustness. Overall, DCDGA shows strong potential as a scalable, high-quality optimisation method, while SGA remains preferable for extremely time-critical scenarios.

References

[1] A. Aziz, W. Osamy, A. M. Khedr, and A. Salim, "Iterative selection and correction based adaptive greedy algorithm for compressive sensing reconstruction," *J. King Saud Univ. – Comput. Inf. Sci.*, vol. 34, no. 3, pp. 892–900, 2022, doi: 10.1016/j.jksuci.2020.03.010.

[2] A. Casado, S. Bermudo, A. D. López-Sánchez, and J. Sánchez-Oro, "An iterated greedy algorithm for finding the minimum dominating set in graphs," *Math. Comput. Simul.*, vol. 207, pp. 41–58, 2023, doi: 10.1016/j.matcom.2022.12.018.

[3] S. Chen, Q. K. Pan, L. Gao, and H. Y. Sang, "A population-based iterated greedy algorithm to minimize total flowtime for the distributed blocking flowshop scheduling problem," *Eng. Appl. Artif. Intell.*, vol. 104, Art. no. 104375, 2021, doi: 10.1016/j.engappai.2021.104375.

[4] R. Duvignau and R. Klasing, "Greediness is not always a vice: Efficient discovery algorithms for assignment problems," *Procedia Comput. Sci.*, vol. 221, pp. 43–52, 2023, doi: 10.1016/j.procs.2023.08.212.

[5] H. M. Fadhil, M. N. Abdullah, and M. I. Younis, "Innovations in t-way test creation based on a hybrid hill climbing-greedy algorithm," *IAES Int. J. Artif. Intell.*, vol. 12, no. 2, pp. 794–805, 2023, doi: 10.11591/ijai.v12.i2.pp794-805.

[6] Z. Gao, J. R. Birge, R. L. Y. Chen, and M. Cheung, "Greedy algorithms for the freight consolidation problem," in *OpenAccess Series in Informatics*, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022, doi: 10.4230/OASIcs.ATMOS.2022.4.

[7] O. Gokalp, E. Tasci, and A. Ugur, "A novel wrapper feature selection algorithm based on iterated greedy metaheuristic for sentiment classification," *Expert Syst. Appl.*, vol. 146, Art. no. 113176, 2020, doi: 10.1016/j.eswa.2020.113176.

[8] S. Gu, G. Shi, W. Wu, and C. Lu, "A fast double greedy algorithm for non-monotone DR-submodular function maximization," *Discrete Math. Algorithms Appl.*, vol. 12, no. 1, Art. no. 2050007, 2020, doi: 10.1142/S179383092050007X.

[9] Ö. İ. Güneri, B. Durmuş, and D. Aydın, *Science Stays True Here*. Science Signpost Publishing, n.d.

[10] Y. Y. Huang, Q. K. Pan, J. P. Huang, P. N. Suganthan, and L. Gao, "An improved iterated greedy algorithm for the distributed assembly permutation flowshop scheduling problem," *Comput. Ind. Eng.*, vol. 152, Art. no. 107021, 2021, doi: 10.1016/j.cie.2020.107021.

[11] K. C. Jha, R. K. Ahuja, and G. Şahin, "New approaches for solving the block-to-train assignment problem," *Networks*, vol. 51, no. 1, pp. 48–62, 2008, doi: 10.1002/net.20195.

[12] T. Kämpfe *et al.*, "Probabilistic greedy algorithm solver using magnetic tunneling junctions for traveling salesman problem," *Res. Square*, 2025, doi: 10.21203/rs.3.rs-5700548/v1.

[13] Y. Kumar, P. Dixit, A. Srivastava, and R. Sahoo, "Investigating optimization methods in computer science engineering: A comprehensive study," in *Lect. Notes Netw. Syst.*, Springer, 2024, pp. 841–852, doi: 10.1007/978-981-97-0641-9_57.

[14] G. Leibovitz and R. Giryes, "Efficient least residual greedy algorithms for sparse recovery," *IEEE Trans. Signal Process.*, vol. 68, pp. 3707–3722, 2020, doi: 10.1109/TSP.2020.2988427.

[15] Y. Z. Li, Q. K. Pan, J. Q. Li, L. Gao, and M. F. Tasgetiren, "An adaptive iterated greedy algorithm for distributed mixed no-idle permutation flowshop scheduling problems," *Swarm Evol. Comput.*, vol. 63, Art. no. 100874, 2021, doi: 10.1016/j.swevo.2021.100874.

[16] J. Q. Li *et al.*, "A hybrid iterated greedy algorithm for a crane transportation flexible job shop problem," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 3, pp. 2153–2170, 2022, doi: 10.1109/TASE.2021.3062979.

[17] D. K. Pradhan, J. Chakraborty, P. Choudhary, and S. Nandi, "An automated conflict of interest based greedy approach for conference paper assignment system," *J. Informetrics*, vol. 14, no. 2, Art. no. 101022, 2020, doi: 10.1016/j.joi.2020.101022.

[18] H. X. Qin *et al.*, "A quick and effective iterated greedy algorithm for energy-efficient hybrid flow shop scheduling problem with blocking constraint," in *Proc. 11th Int. Conf. Inf. Sci. Technol. (ICIST)*, 2021, pp. 325–331, doi: 10.1109/ICIST52614.2021.9440648.

[19] A. Saberi and D. Wajc, "The greedy algorithm is not optimal for on-line edge coloring," in *Leibniz Int. Proc. Informatics (LIPIcs)*, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021, doi: 10.4230/LIPIcs.ICALP.2021.109.

[20] W. Shao, Z. Shao, and D. Pi, "Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem," *Knowl.-Based Syst.*, vol. 194, Art. no. 105527, 2020, doi: 10.1016/j.knosys.2020.105527.

[21] H. S. Shin, T. Li, H. I. Lee, and A. Tsourdos, "Sample greedy based task allocation for multiple robot systems," *Swarm Intell.*, vol. 16, no. 3, pp. 233–260, 2022, doi: 10.1007/s11721-022-00213-0.

[22] E. T. Tetteh and B. Zielosko, "Greedy algorithm for deriving decision rules from decision tree ensembles," *Entropy*, vol. 27, no. 1, Art. no. 35, 2025, doi: 10.3390/e27010035.

[23] C. M. J. Wando and I. Dzikria, "Delivery route estimation on a web-based restaurant delivery system using greedy algorithm," *J. Inf. Technol. Cyber Secur.*, vol. 1, no. 1, pp. 31–40, 2023, doi: 10.30996/jitcs.7611.

[24] M. R. Wayahdi, S. H. N. Ginting, and D. Syahputra, "Greedy, A-Star, and Dijkstra's algorithms in finding shortest path," *Int. J. Adv. Data Inf. Syst.*, vol. 2, no. 1, pp. 45–52, 2021, doi: 10.25008/ijadis.v2i1.1206.

[25] K. W. Ali, R. S. Hawezi, S. W. Kareem, F. S. Khoshabai, and S. K. Askar, "Metaheuristic algorithms in optimization and its application: A review," unpublished manuscript, 2022.

[26] D. Xiang, H. Lin, J. Ouyang, and D. Huang, "Combined improved A* and greedy algorithm for path planning of multi-objective mobile robot," *Sci. Rep.*, vol. 12, no. 1, Art. no. 16310, 2022, doi: 10.1038/s41598-022-17684-0.

[27] K. Yamada *et al.*, "Fast greedy optimization of sensor selection in measurement with correlated noise," *Mech. Syst. Signal Process.*, vol. 158, Art. no. 107619, 2021, doi: 10.1016/j.ymssp.2021.107619.

[28] P. Zhan, "Simultaneous eating algorithm and greedy algorithm in assignment problems," *J. Comb. Optim.*, vol. 45, no. 5, pp. 1482–1500, 2023, doi: 10.1007/s10878-023-01063-0.

[29] Z. Zhao, S. Liu, M. Zhou, D. You, and X. Guo, "Heuristic scheduling of batch production processes based on Petri nets and iterated greedy algorithms," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 1, pp. 251–261, 2022, doi: 10.1109/TASE.2020.3027532.

[30] Z. Y. Zhao, M. C. Zhou, and S. X. Liu, "Iterated greedy algorithms for flow-shop scheduling problems: A tutorial," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 3, pp. 1941–1959, 2022, doi: 10.1109/TASE.2021.3062994.

[31] W. Q. Zou, Q. K. Pan, and M. F. Tasgetiren, "An effective iterated greedy algorithm for solving a multi-compartment AGV scheduling problem in a matrix manufacturing workshop," *Appl. Soft Comput.*, vol. 101, Art. no. 106945, 2021, doi: 10.1016/j.asoc.2020.106945.

[32] Y. Kumar, R. C. Sahoo, and P. Dixit, "A computational analysis of optimization techniques to solve assignment problem," in *Proc. 2nd Int. Conf. Adv. Comput., Commun. Inf. Technol. (ICAICCIT)*, IEEE, 2024, pp. 649–656, doi: 10.1109/ICAICCIT64383.2024.10912276.