Santa Workshop Optimization using Genetic Algorithms

Adagamov Nurmukhammet, Bulat Fakhrutdinov, and Danil Eramasov

***Abstract*—This paper presents a genetic algorithm (GA) approach to solving a constrained scheduling problem where 5000 families must be assigned to 100 days, ensuring daily attendance remains between 125 and 300 individuals. The core contribution of our work lies in the development of a robust initialization method that guarantees valid initial solutions while favoring family preferences in a probabilistic manner. All evolutionary operations—including crossover and mutation—were carefully designed to preserve constraint validity. An elitism strategy was employed to retain top-performing solutions across generations, and tournament selection guided the reproduction process. The cost function was highly optimized using the `numba` library to ensure computational efficiency. Empirical results demonstrate strong performance: the algorithm consistently generates valid initial solutions with a mean cost of 2.5 million, and after 4000 generations, it reduces the cost to below 150,000. This demonstrates the effectiveness of the proposed GA framework under strict feasibility requirements.**

***Index Terms*—Genetic Algorithms, Santa Workshop Scheduling, Optimization, Constraint Satisfaction, Evolutionary Computation**

I. INTRODUCTION

This section provides an overview of the scheduling optimization problem addressed in this work. The task involves assigning 5000 families to 100 available days while maintaining strict bounds on daily attendance. The motivation arises from real-world planning scenarios where constraints such as capacity and preference must be balanced efficiently. The primary objective of this study is to develop a Genetic Algorithm (GA)-based solution capable of producing valid and optimized schedules. Key goals include maintaining feasibility at all stages of the algorithm and achieving significant cost reduction over generations. The proposed method incorporates customized initialization, selection, crossover, and mutation strategies, all designed to respect problem-specific constraints.

II. RELATED WORK

GAs have been widely used in combinatorial optimization tasks due to their flexibility and adaptability. Prior research has explored constraint handling using penalty functions, repair heuristics, and hybrid models. However, many traditional methods struggle to maintain feasibility under strict constraints. Studies such as those based on the Santa Workshop challenge highlight the importance of integrating feasibility checks into the GA process. Our approach builds on these insights by designing a pipeline that ensures all generated solutions remain valid throughout the optimization process.

III. METHODOLOGY

The core idea of our solution is to exclusively operate on **valid solutions** throughout the entire Genetic Algorithm (GA) pipeline. That is, every individual (chromosome) generated during initialization or through mutation/crossover satisfies the problem's strict constraint: the number of people assigned to each of the 100 days must remain within the range of 125 to 300.

A. Chromosome Representation

A custom `Chromosome` class was implemented to encapsulate both the gene representation and the daily occupancy tracking. The chromosome is represented by a NumPy array of length 5000, where each index corresponds to a family, and the value denotes the assigned day. This dual structure allows for efficient updates and quick validation of constraints during the GA operations.

B. Initialization Strategy

The initialization method is designed to always produce valid chromosomes. It does this through a probabilistic and constraint-aware day assignment process:

- Each family's top 10 preferred days (plus one random day) are evaluated.
- Day options are filtered based on capacity constraints — days already at or near 300 are penalized or excluded.
- Each valid option is scored based on:
  - The family's preference rank for that day.
  - The current occupancy of the day.
- One of the valid days is randomly selected with higher probability given to less crowded and more preferred days.

This process ensures:

- Full validity with respect to daily attendance bounds.
- High diversity across the initial population.
- A natural balance between family satisfaction and load balancing.

C. Cost Function Optimization

The cost function evaluates the fitness of a chromosome by computing the penalty based on preference violations and occupancy fluctuation. To ensure high performance during GA evolution, the cost function was optimized using the `numba` library, allowing for just-in-time (JIT) compilation. This resulted in a drastic speedup, bringing evaluation time down to a few milliseconds per chromosome.

D. Selection Strategy

We used tournament selection to choose individuals for the next generation. This balances exploitation and exploration by selecting the best out of a small random subset from the population. This approach maintains selection pressure toward better solutions while avoiding premature convergence.

E. Crossover Strategy

The crossover function creates two new children by combining the gene information from two parents, while ensuring validity throughout:

- Copies of the parent chromosomes are made to preserve the originals.
- Families are processed in a specified or random order.
- For each family, the assigned days in both parents are considered.
- The function checks whether swapping these assignments would:
  - Keep both children's schedules valid.
  - Or at least one child valid (if partial swaps are allowed).
- A swap is performed based on a given probability parameter.

Key features:

- Validity is strictly preserved.
- Partial swaps allow for localized improvement when full swaps aren't feasible.
- The random order option prevents positional bias.

This operator effectively blends genetic material from both parents, while safeguarding feasibility and allowing controlled diversity in the offspring.

F. Mutation Strategy

Mutation introduces small, localized changes that can improve family satisfaction:

- With a probability defined by the mutation rate, one family is selected randomly.
- The algorithm checks whether there exists a better day in the family's preference list that:
  - Is more preferred than the current assignment.
  - Respects the daily occupancy constraint (125–300 people).
- If such a day is found, the family is moved and the daily occupancy updated accordingly.

This method guarantees:

- Validity is preserved at all times.
- Only beneficial mutations (based on family preference) are applied.
- Local optimization is introduced without risking major disruptions.

The mutation strategy is simple yet effective in driving gradual improvement while maintaining feasibility.

G. Evolution Strategy and Elitism

A simple but effective **elitism mechanism** was applied to preserve the top-performing solutions from each generation. Specifically, the top $P\%$ (e.g., 10%) of chromosomes from the current generation, ranked by lowest cost, were directly copied to the next generation. This prevents loss of the best solutions during stochastic operations like crossover and mutation.

H. Reproduction Pipeline

The reproduction process followed a standard GA loop with added validity enforcement at every step. The overall process per generation was:

1) Select a mating pool using tournament selection.
2) Iterate through the pool in pairs:
   - Apply crossover to produce two children.
   - Apply mutation to each child.
   - Ensure both children remain valid.
3) Add elite individuals from the previous generation.
4) Form the new generation by combining elites and newly produced children.

I. Genetic Algorithm Pseudocode

```
FUNCTION genetic_algorithm():
    Initialize population with valid chromosomes
    FOR each generation:
        Evaluate cost of each chromosome
        Select elite chromosomes (top P%)
        Select parents via tournament selection
        Create children via crossover
        Apply mutation to each child
        Form new population: elites + children
        Update best solution if needed
```

J. Hyperparameters and Performance

The GA was configured using the following hyperparameters:

- Population size: `100`
- Number of generations: `4000`
- Tournament size: `5`
- Crossover probability: `0.5`
- Single-swap allowed during crossover: `True`
- Random family processing order: `True`
- Mutation rate: `0.3`
- Elitism ratio: `0.1`

This configuration yielded the following performance results:

- Initial best cost: **1,868,638.2**
- Final best cost: **149,880.1**

This significant improvement confirms the effectiveness of the initialization, crossover, and mutation strategies under constrained optimization.

IV. GitHub Repository

The complete source code, including all experiments and plotting utilities, is publicly available on GitHub Repository :
https://github.com/examplefirstaccount/nic_project

V. Experiments and Evaluation

A. Baseline Approach and Motivation

Our initial attempt involved implementing a standard Genetic Algorithm (GA) without enforcing the daily attendance constraints. While this approach reached a cost of approximately 600,000 after 2000 generations, it consistently produced invalid schedules, making it unsuitable for practical use. This highlighted the need to design our algorithm such that all chromosomes are valid by construction.

B. Initialization Function Experiments

Two initialization strategies were evaluated. The first aimed to greedily fill days to meet minimum occupancy requirements before assigning remaining families. However, it did not prioritize higher-ranked family choices, resulting in poor starting costs (up to 70 million), despite maintaining solution validity.

The second and final initialization approach, described in detail in the Methodology section, achieved a significantly improved average initial cost of approximately 2.5 million (based on 100 runs), while always producing valid chromosomes. This approach balances family preferences with feasibility constraints using a probabilistic penalty-based model.

C. Crossover Testing

The crossover operator was parameterized with:

- Crossover probability p
- Allowance of single-side swaps
- Randomization of family order

To determine optimal configurations, we ran 100 tests across various settings. The results are summarized in Table I.

TABLE I
Crossover Performance Across Settings (100 Runs)

Configuration	Avg Swap Rate	Avg Cost Δ
$p = 1.0$, no swap, ordered	0.9706	-214,258
$p = 1.0$, single swap, ordered	0.9925	+440,343
$p = 1.0$, no swap, random	0.9727	-131,461
$p = 1.0$, single swap, random	**0.9949**	**+525,987**

Results clearly show that enabling single swaps and processing families in random order consistently improved cost. Lowering crossover probability in further tests led to diminished performance. Hence, final settings used were: $p = 0.5$, `allow_single_swap=True`, and `random_order=True`.

D. Mutation Strategy Evaluation

Three mutation strategies were tested:

1) **Random Valid Mutation:** picks one family and makes any valid change
2) **Mass Mutation:** attempts to mutate all families with small probability
3) **Greedy Valid Mutation:** picks one family and applies the best valid improvement (described in Methodology)

Mutation 2 resulted in unstable behavior and poorer convergence. Mutation 1 performed well early on but became too disruptive in later generations. Mutation 3 provided the best overall performance with consistent improvements and stability, although it tended to reduce population diversity and occasionally converged to local optima. Future work may explore introducing occasional ”crazy” or random mutations to escape these optima.

E. Elitism

We applied a simple elitism strategy, carrying over the top 10% of the population unchanged to each new generation. While increasing the elitism rate had little additional benefit, including the mechanism helped stabilize early convergence and preserve high-quality solutions.

F. Hyperparameter Observations

Manual tuning was used to select hyperparameters, summarized below:

- **Population size:** 100 (higher values increased runtime without notable benefit)
- **Generations:** 4000 (longer runs continue improving results)
- **Tournament size:** 5 (values between 3–5 had similar performance)
- **Crossover probability:** 0.5
- **Mutation rate:** 0.3 (higher rates led to faster convergence, but increased risk of stagnation)
- **Elitism ratio:** 0.1

G. Final Performance

With the above configuration, our GA started with a valid solution costing approximately 1.87 million in the first generation:

```
Generation 1: Best Cost = 1868638.2
```

After 4000 generations, the best solution found had a final cost of:

```
Best Cost: 149880.1
```

This represents a significant reduction and demonstrates the effectiveness of our validity-constrained evolutionary approach.

VI. Analysis and Observations

The convergence of the best cost over 4000 generations, as depicted in Figure 1, demonstrates the effectiveness of the optimization algorithm in minimizing the cost function. Key observations from the graph include:

Initial Rapid Convergence: The algorithm exhibits a steep decline in the best cost during the early generations, indicating efficient exploration of the solution space (due to a variety of solutions) and rapid identification of promising regions.

Stabilization Phase: After the initial rapid improvement, the rate of convergence slows, suggesting that the algorithm transitions from exploration to exploitation, fine-tuning solutions to achieve marginal gains.

Final Convergence: The best cost plateaus in later generations, implying that the algorithm has likely reached a

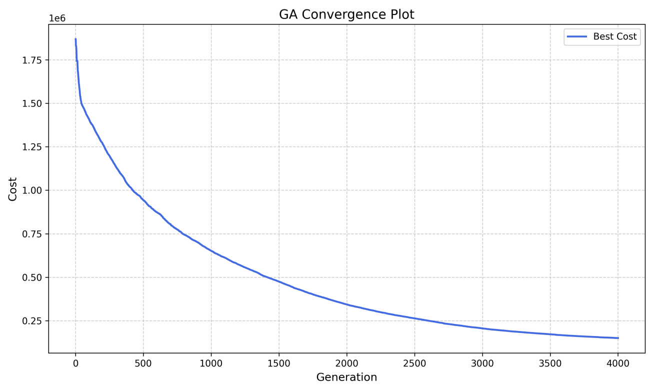

Fig. 1. Convergence of best cost over 4000 generations.

near-optimal or local optimum solution, with further iterations yielding negligible improvements.

Algorithm Robustness: The smoothness of the convergence curve indicates stable performance, free from abrupt fluctuations, which underscores the robustness of the optimization process.

These observations highlight the algorithm's capability to balance exploration and exploitation, ensuring effective convergence toward an optimal solution.

VII. Conclusion

By enforcing feasibility at every step of the evolutionary cycle, the algorithm consistently produces valid and optimized solutions. Contributions include a constraint-aware initialization method, a mutation strategy that preserves validity, and the use of elitism and tournament selection to guide convergence. Empirical evaluation demonstrates a significant cost reduction from the initial to the final generation. Future directions may include introducing diversity-enhancing strategies or hybrid approaches to avoid local optima and further improve solution quality.

Acknowledgment

The authors would like to thank each other for shared effort, debugging marathons, and pizza-fueled commits.

References

[1] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
[2] Kaggle Santa Workshop 2019: https://www.kaggle.com/c/santa-workshop-tour-2019