UNIVERSITY OF BIRMINGHAM

Comparative Metaheuristic Strategies for Airline Crew Scheduling

A Study of Simulated Annealing and Genetic Algorithm Approaches

Evolutionary Computation

Author: Max Hart

mah422@student.bham.ac.uk

Professors: Dr Shan He & Dr Per Kristian Lehre

March 4, 2025

Contents

1 Introduction

The airline crew scheduling problem, commonly modelled as a Set Partitioning Problem (SPP), requires assigning each flight leg to exactly one valid crew rotation without violating complex operational constraints. Exact methods often struggle to scale, prompting the use of metaheuristics, which can navigate vast solution spaces effectively while balancing cost and constraint satisfaction.

In this report, we present and evaluate three such algorithms developed from scratch:

- **Simulated Annealing (SA)**: A straightforward local search guided by a probabilistic acceptance rule.
- **Standard Binary Genetic Algorithm (BGA)**: An evolutionary approach using selection, crossover, and mutation.
- **Improved BGA**: An enhanced version incorporating pseudo-random initialisation, adaptive mutation, and heuristic repair.

We apply each algorithm to three OR-Library benchmarks (`sppnw41`, `sppnw42`, `sppnw43`), running each 30 times to compare feasibility rates, solution quality, and runtime. Additionally, we examine two methods for constraint handling—ranking replacement and stochastic ranking—to reveal how different strategies affect performance. Our aim is to highlight the practical considerations in airline crew scheduling and identify which metaheuristic features yield the best results under varying problem complexities.

2 Metaheuristic Algorithms for the SPP

In this section, we will provide a comprehensive overview of the three metaheuristic algorithms developed for solving the SPP. Each algorithm has been implemented from scratch in Python and is designed to find high-quality solutions to the airline crew scheduling problem. For each algorithm, we will provide a brief introduction outlining its underlying principles and technical merits, followed by the corresponding pseudocode. Detailed flowcharts will be presented on subsequent pages to visually encapsulate the step-by-step processes. This thorough explanation is intended to clarify both the operational mechanics and the innovative aspects of our methods.

2.1 Simulated Annealing

Simulated Annealing is a local search technique that iteratively improves a candidate solution by exploring its neighbourhood. It uses a temperature parameter to probabilistically accept inferior solutions, thereby enabling the algorithm to escape local optima. In each iteration, a small random modification (or "bit-flip" in the SPP context) generates a neighbouring solution, which is then evaluated using a penalty-based cost function. Even if the neighbour is worse, it may still be accepted with a probability tied to the current temperature, allowing for occasional uphill moves that help the search avoid getting trapped in local minima.

2.1.1 Pseudocode for Simulated Annealing

The pseudocode below outlines the Simulated Annealing algorithm for the SPP, detailing random solution initialization, neighbour generation via bit-flipping, acceptance via direct improvement or the Metropolis criterion, and temperature cooling.

Algorithm 1 SimulatedAnnealing(T, α, maxIter, penaltyFactor)

Initialisation:
$x \leftarrow$ RandomSolution() ▷ Generate a random binary solution
$F(x) \leftarrow$ PenaltyFitness(x, penaltyFactor) ▷ Cost + coverage violations * penalty
$x_{\text{best}} \leftarrow x$; $F_{\text{best}} \leftarrow F(x)$
for $iter = 1$ to maxIter **do**
    Neighbour generation:
    $x' \leftarrow$ FlipOneRandomBit(x) ▷ Create neighbour by flipping exactly one bit
    Evaluate neighbour:
    $F(x') \leftarrow$ PenaltyFitness(x', penaltyFactor)
    if $F(x') < F(x)$ **then**
        $x \leftarrow x'$ ▷ If neighbour is better, accept it outright
    else
        $\Delta \leftarrow F(x') - F(x)$ ▷ Compute the increase in fitness
        Metropolis criterion:
        if rand() $< e^{-\Delta/T}$ **then**
            $x \leftarrow x'$ ▷ Accept worse solution with probability $e^{-\Delta/T}$
    Update global best:
    if $F(x) < F_{\text{best}}$ **then**
        $x_{\text{best}} \leftarrow x$; $F_{\text{best}} \leftarrow F(x)$
    Cool down:
    $T \leftarrow \alpha \times T$
return $(x_{\text{best}}, F_{\text{best}})$

2.1.2 Flowchart for Simulated Annealing

The flowchart visually represents the step-by-step process of the Simulated Annealing algorithm, from initialisation and neighbour evaluation to acceptance decisions and temperature cooling.

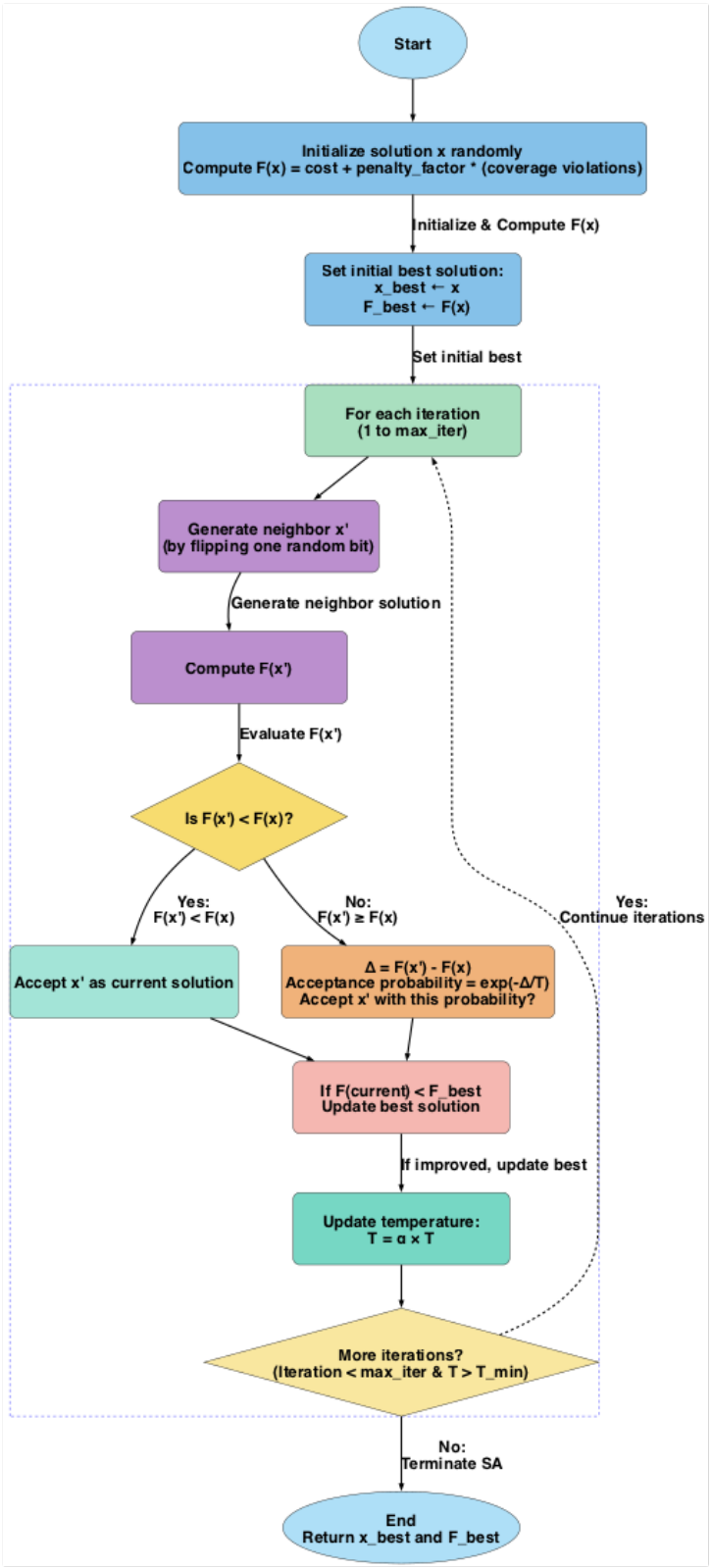

Figure 1: Flowchart of the Simulated Annealing Algorithm for the Set Partitioning Problem.

2.2 Standard Binary Genetic Algorithm

The Standard Binary Genetic Algorithm is a population-based technique that evolves a set of candidate solutions using selection, crossover, and mutation operators. It utilises tournament selection to choose parents, applies one-point crossover to combine their genetic material, and uses bit-flip mutation to introduce variability. A penalty function is employed to combine the cost of a solution with constraint violations, ensuring that feasibility is encouraged over generations. This method is well-suited for problems with discrete binary representations and offers robustness through its stochastic operations.

2.2.1 Pseudocode for Standard BGA

The pseudocode below outlines the Standard Binary Genetic Algorithm for the SPP, detailing population initialisation, tournament selection, crossover, mutation, and population update.

Algorithm 2 StandardBGA(*popSize*, *cxRate*, *mutRate*, *maxGens*, *penaltyFactor*, *tournK*)

Initialise:
$P \leftarrow$ RandomPopulation(*popSize*) ▷ Generate *popSize* random binary solutions
EvaluateFitness(P, *penaltyFactor*) ▷ Compute cost + penalty for each individual
$(xBest, F_{\text{best}}) \leftarrow$ BestSolution(P) ▷ Track best overall
for $g = 1 \rightarrow maxGens$ **do**
    $Q \leftarrow \varnothing$ ▷ Offspring population
    while $|Q| < popSize$ **do**
        $p_1 \leftarrow$ TournamentSelect(P, *tournK*) ▷ Select parent 1
        $p_2 \leftarrow$ TournamentSelect(P, *tournK*) ▷ Select parent 2
        if rand() $<$ *cxRate* **then**
            $(c_1, c_2) \leftarrow$ OnePointCrossover(p_1, p_2) ▷ Split at random point and swap tails
        else
            $c_1 \leftarrow p_1, \quad c_2 \leftarrow p_2$ ▷ No crossover; children are copies
        Mutate(c_1, *mutRate*)
        Mutate(c_2, *mutRate*) ▷ Bit-flip mutation with prob = *mutRate*
        EvaluateFitness($\{c_1, c_2\}$, *penaltyFactor*) ▷ Compute cost + penalty of each child
        $Q \leftarrow Q \cup \{ c_1, c_2 \}$
    $P \leftarrow Q$ ▷ Replace old population with offspring
    $(xCurrBest, F_{\text{currBest}}) \leftarrow$ BestSolution(P)
    if $F_{\text{currBest}} < F_{\text{best}}$ **then**
        $xBest \leftarrow xCurrBest, \quad F_{\text{best}} \leftarrow F_{\text{currBest}}$ ▷ Update global best if improved
return $(xBest, F_{\text{best}})$ ▷ Best solution found

2.2.2 Flowchart for Standard BGA

The flowchart provides a visual overview of the Standard BGA's iterative process, from offspring generation to re-ranking and best solution update.

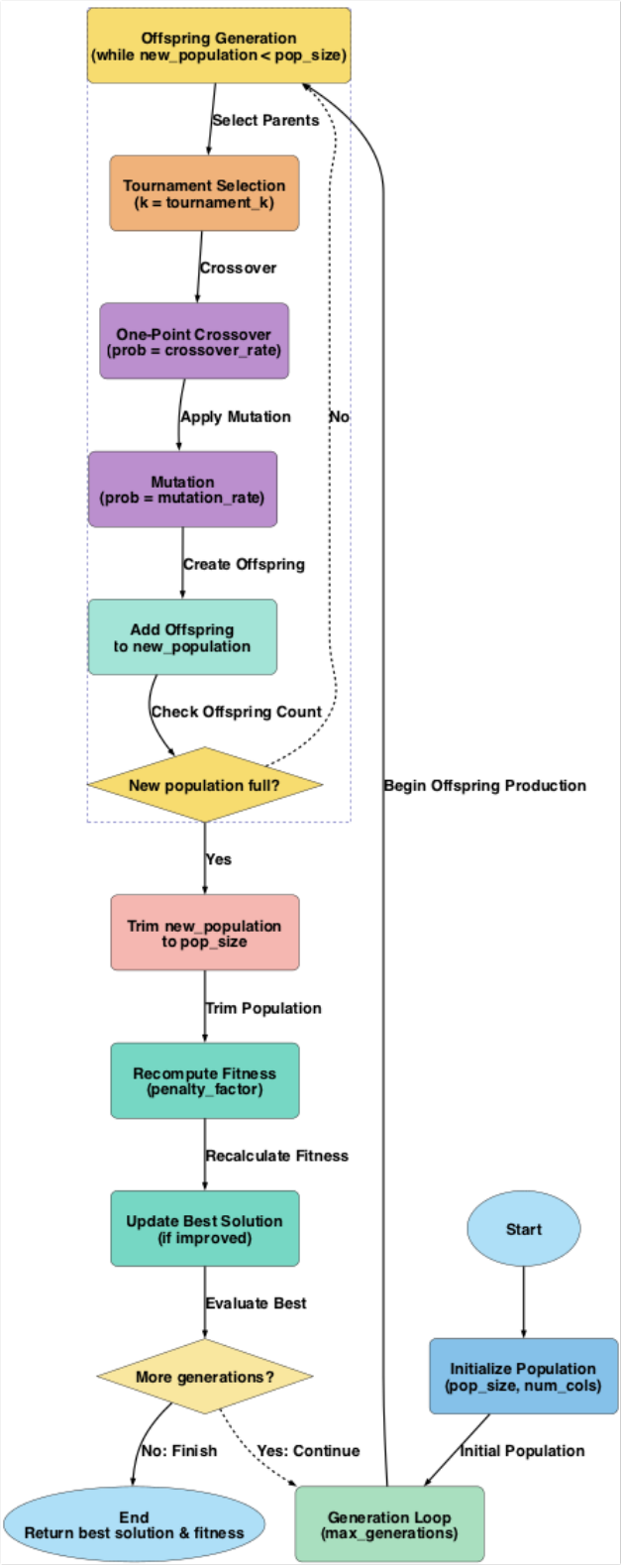

Figure 2: Flowchart of the Standard Binary Genetic Algorithm for the Set Partitioning Problem.

2.3 Improved Binary Genetic Algorithm

The Improved Binary Genetic Algorithm extends the standard Binary Genetic Algorithm by integrating several problem-specific enhancements. It begins with a pseudo–random initialisation that aims to reduce over–coverage in the initial population. The algorithm then incorporates a stochastic ranking procedure that probabilistically balances cost and constraint violation, allowing for a more flexible handling of infeasible solutions. Adaptive mutation is employed to reintroduce promising genetic material, and a DROP/ADD heuristic improvement operator repairs infeasible solutions. These enhancements, inspired by Chu & Beasley [1] and Runarsson & Yao [2], make the Improved BGA particularly robust for the SPP. Overall, this algorithm offers a more sophisticated search mechanism by blending global search operators with domain-specific repairs.

2.3.1 Pseudocode for Improved BGA

The pseudocode below describes the Improved Binary Genetic Algorithm for the SPP, integrating pseudo–random initialisation, stochastic ranking, adaptive mutation, and heuristic repair.

Algorithm 3 ImprovedBGA(*popSize*, *maxGens*, p_{stoch}, ...)

Initialise:
$P \leftarrow$ PseudoRandomInit(*popSize*) ▷ Chu & Beasley [1] for partial coverage
EvaluateCostUnfitness(P) ▷ For each individual, compute (cost, violations)
for $g = 1 \rightarrow$ *maxGens* **do**
    StochasticRankSort(P, p_{stoch}) ▷ Runarsson & Yao [2] sort by cost vs. unfitness
    $O \leftarrow \varnothing$ ▷ Offspring population
    while $|O| <$ *popSize* **do**
        $(p_1, p_2) \leftarrow$ SelectParents(P) ▷ Randomly pick two parents
        $(c_1, c_2) \leftarrow$ UniformCrossover(p_1, p_2) ▷ Exchange bits at random (50% each)
        AdaptiveMutation(c_1, P) ▷ Bit-flip mutation + forced coverage
        AdaptiveMutation(c_2, P)
        HeuristicImprove(c_1) ▷ Repair via DROP/ADD (Chu & Beasley [1])
        HeuristicImprove(c_2)
        EvaluateCostUnfitness($\{c_1, c_2\}$)
        $O \leftarrow O \cup \{c_1, c_2\}$
    $C \leftarrow P \cup O$ ▷ Combine parents & offspring
    StochasticRankSort(C, p_{stoch}) ▷ Re-sort combined pop
    $P \leftarrow$ Top(C, *popSize*) ▷ Keep the best *popSize*
return BestFeasible(P) ▷ Return best feasible (lowest cost, unfitness=0)

2.3.2 Flowchart for Improved BGA

The flowchart visually maps the Enhanced BGA's process, clearly highlighting its advanced offspring production and repair mechanisms.

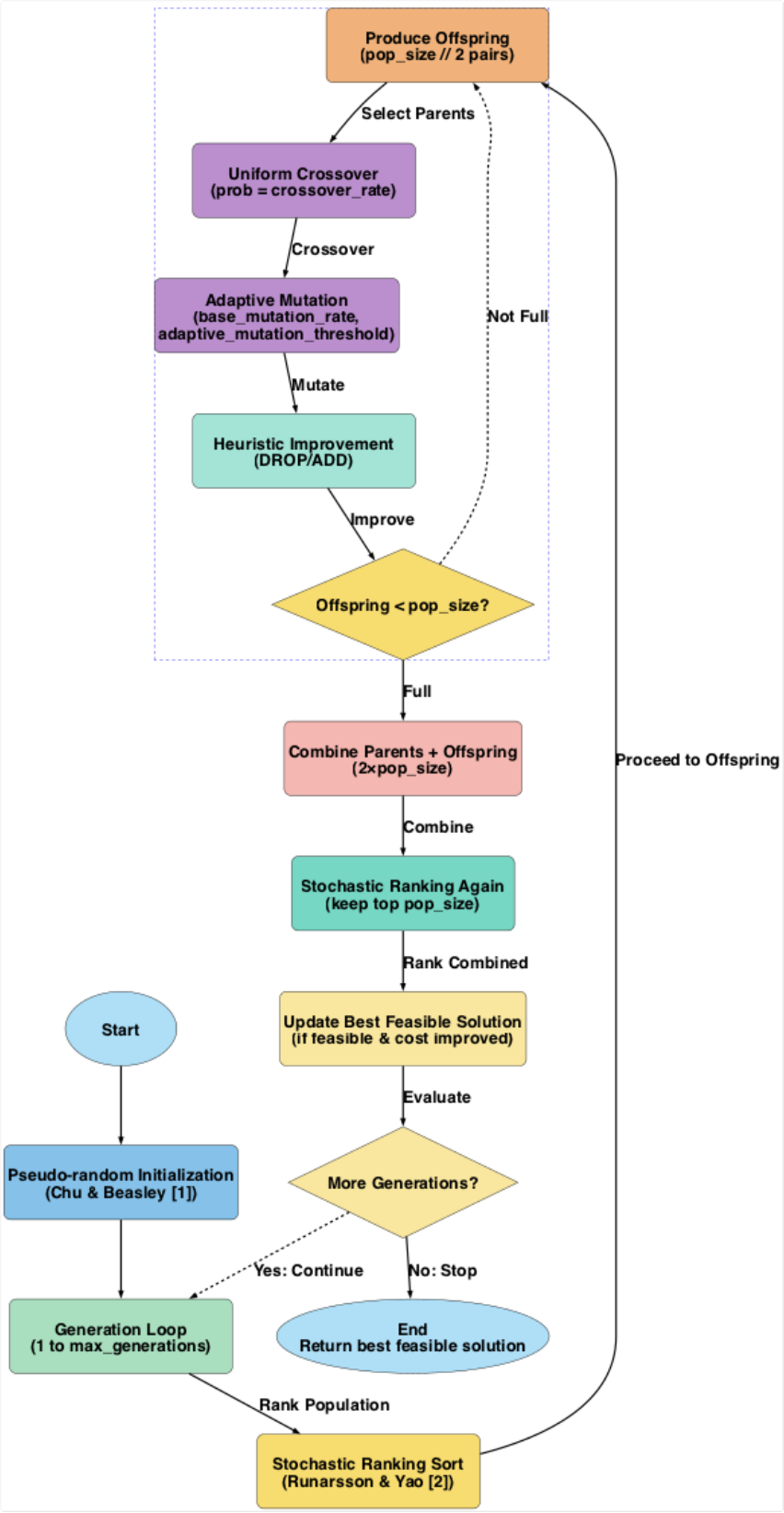

Figure 3: Flowchart of the Improved Binary Genetic Algorithm for the Set Partitioning Problem.

3 Benchmark Results

In this section, we present the results obtained by our three algorithms—Simulated Annealing (SA), Standard BGA, and Improved BGA—on three different problem sets. Each subsection provides a table summarising performance metrics across 30 independent runs, focusing on feasibility and overall solution quality. In particular, we report the rate at which each method produces a feasible solution, best/worst feasible cost, average feasible cost (with standard deviation), and average execution time (with standard deviation).

3.1 Problem Set `sppnw41`

We tested each algorithm for 30 independent runs on the `sppnw41` problem instance. Table 3.1 shows the feasibility rate, along with best, worst, and mean feasible costs (plus standard deviation), as well as average run time (plus standard deviation) for each algorithm. All results are based on runs that reached feasibility (zero coverage violations). A brief description of the hyperparameters used in each method can be found below.

Algorithm	Feas. Rate	Best Cost	Worst Cost	Mean Cost	Std. Dev.	Time (s) ± SD
SA	12/30	18,066	32,730	23,344	4,002	0.082 ± 0.0008
Standard BGA	21/30	12,678	28,269	20,188	3,736	1.94 ± 0.07
Improved BGA	30/30	11,307	11,676	11,319	66.2	1.18 ± 0.06

Table 1: Performance on `sppnw41` (30 runs). Feasible solutions have zero coverage violations.

Hyperparameters:

- **Simulated Annealing (SA).** Initial temperature = 2000, cooling factor $\alpha = 0.99$, maximum iterations = $10,000$, penalty factor = $50,000$.
- **Standard BGA.** Population size = 250, crossover rate = 0.8, mutation rate = 0.003, max generations = 500, penalty factor = 4800, tournament size = 4.
- **Improved BGA.** Population size = 20, max generations = 400, crossover rate = 0.85, base mutation rate = 0.03, stochastic rank $p = 0.45$, adaptive mutation threshold = 0.2, adaptive mutation count = 2.

As shown, the Improved BGA achieves a 100% feasibility rate and outperforms the other algorithms in terms of average feasible cost for this problem set. The Standard BGA also performs well, with a high feasibility rate and competitive costs. Simulated Annealing, while less consistent, still manages to find feasible solutions in 40% of runs.

3.2 Problem Set `sppnw42`

We tested each algorithm for 30 independent runs on the `sppnw42` problem instance. Table 3.2 shows the feasibility rate, along with best, worst, and mean feasible costs (plus standard deviation),

as well as average run time (plus standard deviation) for each algorithm. All results are based on runs that reached feasibility (zero coverage violations). A brief description of the hyperparameters used in each method can be found below.

Algorithm	Feas. Rate	Best Cost	Worst Cost	Mean Cost	Std. Dev.	Time (s) ± SD
SA	1/30	15,072	15,072	15,072	0	0.59 ± 0.000
Standard BGA	0/30	—	—	—	—	—
Improved BGA	30/30	7,666	8,944	7,869	331	10.53 ± 0.28

Table 2: Performance on `sppnw42` (30 runs). Feasible solutions have zero coverage violations.

Hyperparameters:

- **Simulated Annealing (SA).** Initial temperature = 5000.0, cooling factor $\alpha = 0.99$, max iterations = 10,000, penalty factor = 100,000.
- **Standard BGA.** Population size = 400, crossover rate = 0.9, mutation rate = 0.003, max generations = 1000, penalty factor = 10,000, tournament size = 5.
- **Improved BGA.** Population size = 20, max generations = 400, crossover rate = 0.85, base mutation rate = 0.03, stochastic rank $p = 0.45$, adaptive mutation threshold = 0.25, adaptive mutation count = 2.

As shown, the Standard BGA failed to reach feasibility in any run, while Simulated Annealing found a valid solution only once. By contrast, the Improved BGA maintained a 100% feasibility rate and delivered competitive costs consistently.

3.3 Problem Set `sppnw43`

We tested each algorithm for 30 independent runs on the `sppnw43` problem instance. Table 3.3 shows the feasibility rate, along with best, worst, and mean feasible costs (plus standard deviation), as well as average run time (plus standard deviation) for each algorithm. All results are based on runs that reached feasibility (zero coverage violations). A brief description of the hyperparameters used in each method can be found below.

Algorithm	Feas. Rate	Best Cost	Worst Cost	Mean Cost	Std. Dev.	Time (s) ± SD
SA	5/30	11,996	17,614	14,600	2,100	0.57 ± 0.01
Standard BGA	0/30	—	—	—	—	—
Improved BGA	30/30	8,904	10,282	9,528	373	11.45 ± 0.33

Table 3: Performance on `sppnw43` (30 runs). Feasible solutions have zero coverage violations.

Hyperparameters:

- **Simulated Annealing (SA).** Initial temperature = 5000, cooling factor $\alpha = 0.99$, max iterations = 10,000, penalty factor = 100,000.
- **Standard BGA.** Population size = 400, crossover rate = 0.9, mutation rate = 0.003, max generations = 1000, penalty factor = 10,000, tournament size = 5.
- **Improved BGA.** Population size = 20, max generations = 400, crossover rate = 0.85, base mutation rate = 0.03, stochastic rank $p = 0.56$, adaptive mutation threshold = 0.25, adaptive mutation count = 2.

As indicated in Table 3.3, the Standard BGA again failed to produce any feasible solution, whereas Simulated Annealing identified a valid solution in only 16.7% of runs. By contrast, the Improved BGA achieved a 100% feasibility rate with a notably lower mean feasible cost.

4 Comparison of Results

In this section, we compare the performance of Simulated Annealing (SA), the Standard Binary Genetic Algorithm (BGA), and the Improved BGA across three benchmark problem sets (`sppnw41`, `sppnw42`, and `sppnw43`). Our comparison focuses on feasibility (the percentage of runs that yield a valid solution), the quality of feasible solutions (best, worst, and mean feasible cost), and runtime metrics (mean execution time). We also reflect on key hyperparameters that potentially explain differences in performance.

4.1 Feasibility and Solution Quality

Overall, the Improved BGA demonstrates the most robust performance across all three problem sets. It consistently achieves a 100% feasibility rate, indicating that it almost never fails to produce a valid set-partitioning solution. In problem instances `sppnw42` and `sppnw43`, the Improved BGA is essentially the only method obtaining feasible solutions in every run, whereas SA and the Standard BGA sometimes or frequently fail to produce any feasible solutions. The reliability of the Improved BGA can be attributed to several design elements: (1) *pseudo-random initialization* avoids severe over-coverage from the outset, (2) *heuristic improvement (DROP/ADD)* quickly repairs infeasible solutions, and (3) *adaptive mutation* reduces recurring coverage violations by focusing on frequently violated rows.

Focusing on specific metrics in `sppnw41`, the Improved BGA achieves a 100% feasibility rate with a best feasible cost of 11,307 and a mean feasible cost of approximately 11,319. By comparison, SA finds feasible solutions only 40% of the time, and although its best feasible cost is as low as 18,066, the mean feasible cost among runs that succeed is higher (around 23,344). The Standard BGA, with a 70% feasibility rate in `sppnw41`, hits a best feasible cost of 12,678 and an average of 20,188 for feasible runs.

The divergence becomes more pronounced in `sppnw42` and `sppnw43`, where the Standard BGA produces no feasible solutions in 30 runs, while SA only finds a valid solution once (in `sppnw42`) or in 5 out of 30 runs (in `sppnw43`). By contrast, the Improved BGA again remains at 100%

feasibility, maintaining mean feasible costs in the ranges of roughly 7,869–9,528. These outcomes highlight the importance of additional constraint-handling components (such as heuristic repair and adaptive row coverage) present in the Improved BGA but absent or less elaborated in the Standard BGA.

4.2 Runtime and Hyperparameter Influence

Regarding runtime, SA typically completes its search in under a second for `sppnw41` (mean of around 0.082 s) and stays below a second or two in the other problem sets. The BGAs, by virtue of their population-based operations, require longer: for example, the Standard BGA can take over 30 seconds in some runs of `sppnw42` and `sppnw43`, and the Improved BGA often ranges from a few seconds up to tens of seconds. Despite this higher computational cost, the Improved BGA's advanced operators almost guarantee feasibility and competitive costs.

Hyperparameter choices also play a role in these outcomes. For instance, SA employs a relatively large penalty factor to discourage infeasible columns, but the success rate is heavily influenced by the cooling schedule (initial temperature and alpha). The Standard BGA's population size, crossover rate, and mutation rate may be effective in moderately constrained SPPs (such as `sppnw41`) but fall short for larger or more difficult sets. The Improved BGA, on the other hand, leverages a smaller population plus specialized genetic operators (stochastic ranking, adaptive mutation threshold) that prioritize feasibility more effectively.

In summary, although SA has rapid run times, it offers only intermittent feasibility success, and the Standard BGA can excel in simpler instances but may fail completely in more constrained problems. The Improved BGA, while more computationally demanding, consistently yields feasible solutions with the lowest mean cost across all benchmarks, underscoring the benefit of domain-specific heuristics and adaptive strategies in set-partitioning optimisations.

5 Ranking Replacement vs. Stochastic Ranking

Both methods aim to balance cost and constraint satisfaction in the SPP, yet they differ in approach. The **Ranking Replacement** method (Chu & Beasley, 1998) partitions the population deterministically into four subgroups based on fitness (cost) and unfitness (constraint violation). A new solution replaces an individual from the first non-empty subgroup (starting with those worst in both criteria), thereby steadily improving the overall population. This approach offers a clear structure but can be rigid as the balance between cost and constraint violation may evolve during the search.

In contrast, the **Stochastic Ranking** method (Runarsson & Yao, 2000) uses a bubble-sort-like procedure where adjacent solutions are compared probabilistically—if at least one solution is infeasible, they are compared by cost with a set probability P (typically less than 0.5) and by unfitness otherwise. This allows for a more adaptive balance, enabling the search to explore infeasible regions as bridges between isolated feasible areas without rigid subgroup thresholds.

In summary, while both methods share the goal of guiding the search toward feasible, high-quality solutions, Ranking Replacement enforces a strict hierarchical structure, whereas Stochastic Ranking offers a flexible, adaptive mechanism that reduces the need for extensive parameter tuning.

6 Conclusion

In this report, we have examined three metaheuristic approaches—Simulated Annealing, a Standard Binary Genetic Algorithm, and an Improved BGA—for solving airline crew scheduling within the Set Partitioning Problem framework. Our findings demonstrate that while Simulated Annealing can be fast, it is prone to low feasibility rates under tighter constraints. The Standard BGA, meanwhile, can excel on smaller or less constrained problems but may fail altogether for more difficult instances. By contrast, the Improved BGA consistently achieves near-perfect feasibility across all benchmarks, providing robust solutions with low cost through specialised operators such as pseudo-random initialisation, heuristic repair, and stochastic ranking. These results highlight the importance of domain-specific enhancements and adaptive strategies for reliably navigating complex, highly constrained optimisation landscapes.

References

[1] P. C. Chu and J. E. Beasley, *Constraint Handling in Genetic Algorithms: The Set Partitioning Problem*, Journal of Heuristics, 11:323–357, 1998.

[2] T. P. Runarsson and X. Yao, *Stochastic Ranking for Constrained Evolutionary Optimisation*, IEEE Transactions on Evolutionary Computation, 4(3):284–294, 2000.