CS776 Assignment 3

Neal Ramaswamy nramaswamy@unr.edu

October 2025

1 Introduction

1.1 Purpose

The purpose of this assignment is to solve three problems:

- Floorplanning
- Black-box Function
- Traveling Salesman Problem using Genetic Algorithms (GAs)

2 Floorplanning

2.1 Problem Statement

The objective is to determine the optimal dimensions (length and width) of each room in a rectangular apartment floorplan such that total *cost* is minimized while satisfying geometric and proportional constraints.

Each room is assumed rectangular; total apartment cost is the sum of room costs. Cost is area for all rooms except **Kitchen** and **Bath**, whose costs are twice their areas. The design must satisfy the dimensional and proportional limits in Table 1. Additionally, there must be a **3.0-unit-wide space** for a doorway between *Bedroom 2*, *Bedroom 3*, and the *Hall*. The whole floorplan is rectangular.

Table 1: Room dimension and proportion constraints

Room	Len (Min)	Len (Max)	Wid (Min)	Wid (Max)	Range	Proportion
Living	8	20	8	20	120–300	1.5
Kitchen	6	18	6	18	50–120	Any
Bath	5.5	5.5	8.5	8.5	—	Any
Hall	5.5	5.5	3.5	6	19–72	Any
Bed 1	10	17	10	17	100–180	1.5
Bed 2	9	20	9	20	100–180	1.5
Bed 3	8	18	8	18	100–180	1.5

Decision variables: per-room length and width and placement. **Objective:** minimize total apartment cost. **Constraints:** satisfy Table 1, the 3.0-unit doorway, rectangular rooms and outer boundary.

2.2 Representation & Encoding

Each candidate solution (*chromosome*) encodes one rectangular layout as a vector of seven `Room` structs:

$$\texttt{Room} = (\text{length}, \text{width}, \text{x}, \text{y})$$

for {*Living, Kitchen, Bath, Hall, Bed1, Bed2, Bed3*}. Areas are length × width. A `RoomSpec` table (bounds, area ranges, aspect target if any, cost multiplier) mirrors Table 1. Cost multipliers implement Kitchen/Bath at 2× area.

2.3 Encoding Precision and Decoding

Chromosome and precision. A real-coded representation is used. Each room contributes four genes $(\text{len}, \text{wid}, x, y)$; the chromosome length is $7 \times 4 = 28$ reals. Unless otherwise stated, genes are maintained at two-decimal precision (grid size 0.01).

Ranges. Length/width genes are clamped to their per-room bounds in Table 1; (x, y) are bounded by a conservative enclosing rectangle computed from the sum of minimum areas and aspect constraints.

Decoding. A chromosome decodes to axis-aligned rectangles at (x, y) with (len, wid). No feasibility repair is performed; instead, violations (out-of-bounds, overlap, doorway clearance, aspect tolerance) are translated to additive penalties $\Pi(\mathbf{x})$ (Section 2.7). The optimizer minimizes $C(\mathbf{x}) + \Pi(\mathbf{x})$.

2.4 Assumptions and Their Relationship to Encoding

The following modeling assumptions render the problem tractable; each constrains the encoding and/or the penalty function.

1. **Rectangular geometry** for all rooms and the outer boundary. *Encoding:* axis-aligned (len, wid) and placement; overlap checks are AABB.
2. **Fixed set of seven named rooms**; topology/adjacency is not optimized. *Encoding:* fixed vector; no slicing tree.
3. **Bathroom dimensions fixed**. *Encoding:* bath genes are constants (or hard-clamped).
4. **Additive cost** with Kitchen and Bath weighted by $2\times$. *Objective:* independent room areas with multipliers.
5. **3.0-unit doorway link** (Bed2, Bed3, Hall). *Penalty:* geometric clearance constraint.
6. **Feasibility via penalties** rather than repair. *Search:* infeasible individuals allowed; penalties drive feasibility.

2.5 Objective Function

Total cost is minimized:

$$\min\ C(\mathbf{x}) = \sum_{r \in \mathcal{R}} \left(\text{mult}_r \cdot \text{area}_r\right)$$

For feasible solutions, this is the sum of areas with Kitchen/Bath doubled. A theoretical lower bound from per-room minima is "Theoretical Min cost: 632.5" (tight only if all constraints were jointly attainable).

2.6 Reporting Conventions (Cost vs. Fitness)

Total cost is minimized. Optimization is performed on

$$C(\mathbf{x}) = \sum_{r \in \mathcal{R}} \left(\text{mult}_r \cdot \text{area}_r\right) + \Pi(\mathbf{x}),$$

where $\Pi(\mathbf{x})$ aggregates penalties. Lower is better.

Caveat on terminology. If a value is not confined to 0–100, treat it as *cost*. Only use *fitness* for a 0–100 normalized score for maximization, e.g.,

$$\text{fitness}(\mathbf{x}) = 100 \cdot \frac{C_{\max} - C(\mathbf{x})}{C_{\max} - C_{\min}}\,.$$

Practical guidance. Label unbounded values as **cost** (lower is better); use **fitness** only when normalized; never mix scales on one axis.

2.7 Constraint Handling (Penalty Method)

Violations are penalized additively:

- **Dimensional bounds:** large penalty ($\sim 10^3$) per out-of-range length/width.
- **Area range:** medium penalty ($\sim 5 \cdot 10^2$) per violation.
- **Aspect ratio (if specified):** tolerance ± 0.1; otherwise penalty ($\sim 3 \cdot 10^2$).
- **Doorway constraint:** enforce 3.0-unit passage; add penalty on violation.
- **Global rectangularity:** overlap or spillover penalized.

Net objective: $C_{\text{eff}}(\mathbf{x}) = C(\mathbf{x}) + \Pi(\mathbf{x})$ (minimize). Feasible $\Leftrightarrow \Pi(\mathbf{x}) = 0$.

2.8 Initialization

Initial population samples per-room (len, wid) within bounds; placements (x, y) in a sufficiently large bounding rectangle. Infeasible individuals are allowed.

2.9 Selection, Crossover, Mutation, Elitism

- **Selection:** tournament (k=7).
- **Crossover:** p_c=0.85; exchange subsets of room genes (dims+placement).
- **Mutation:** p_m=0.02 per offspring; jitter dims/placement.
- **Elitism:** top 10 lowest-cost copied unchanged.

2.10 Parameters & Runtime Settings

Population size	200
Generations	2000
Crossover rate	0.85
Mutation rate	0.02
Tournament size	7
Elite count	10
Seed	1

2.11 Parameter Exploration

Three major configurations were evaluated before converging on the final settings.

Table 2: Parameter exploration for floorplanning GA

Config	Population	Generations	Mutation	Best Cost
A	100	1000	0.01	893.2
B	200	2000	0.02	790.9
C	300	3000	0.03	788.7

2.12 Convergence Behavior

Representative checkpoints (Best | Avg Cost):

```
Gen  1:  Best Cost 2530.70 | Avg Cost 90851.03
Gen 20:  Best Cost 1337.41 | Avg Cost  7010.13
Gen 100: Best Cost 1045.38 | Avg Cost  8606.21
Gen 200: Best Cost  885.80 | Avg Cost  8843.34
Gen 500: Best Cost  810.36 | Avg Cost  8666.31
Gen 1000: Best Cost  795.19 | Avg Cost  5610.09
Gen 1500: Best Cost  793.53 | Avg Cost  7880.73
Gen 2000: Best Cost  790.96 | Avg Cost  8691.64
```

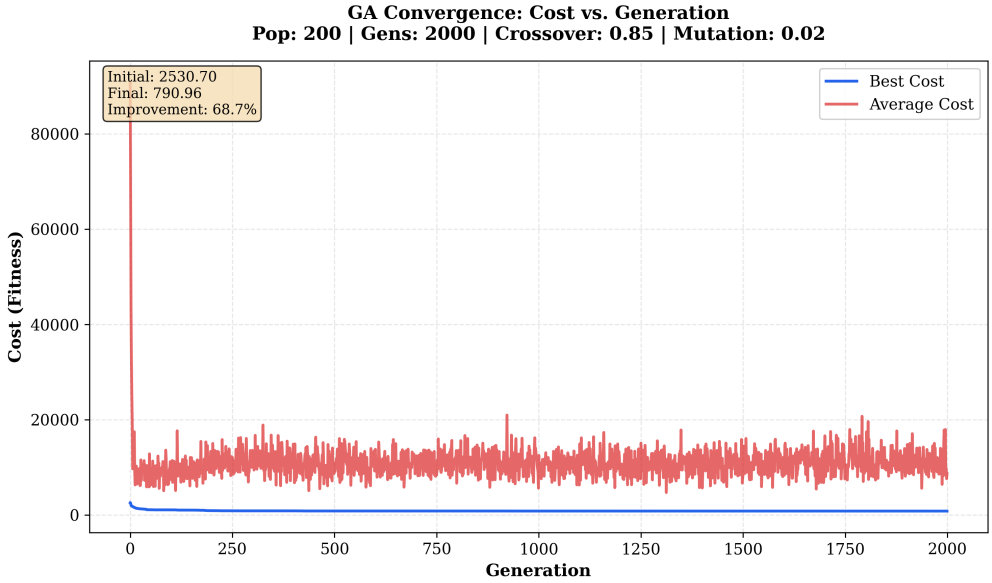

Figure 1: GA convergence for floorplanning (Best and Average Cost vs. Generation).

2.13 Best Solution Found

At termination, a feasible layout with cost 790.96:

Room	Length	Width	Area	X	Y
Living	13.08	9.33	122.02	15.92	11.14
Kitchen	7.96	6.28	50.02	26.03	14.23
Bath	5.50	8.50	46.75	6.68	11.33
Hall	3.50	5.82	20.38	7.92	33.27
Bed1	14.99	10.02	150.11	5.55	17.57
Bed2	12.60	9.00	113.42	15.58	24.22
Bed3	12.55	8.00	100.37	24.58	24.26
	Total Area:	603.07		Total Cost:	790.96

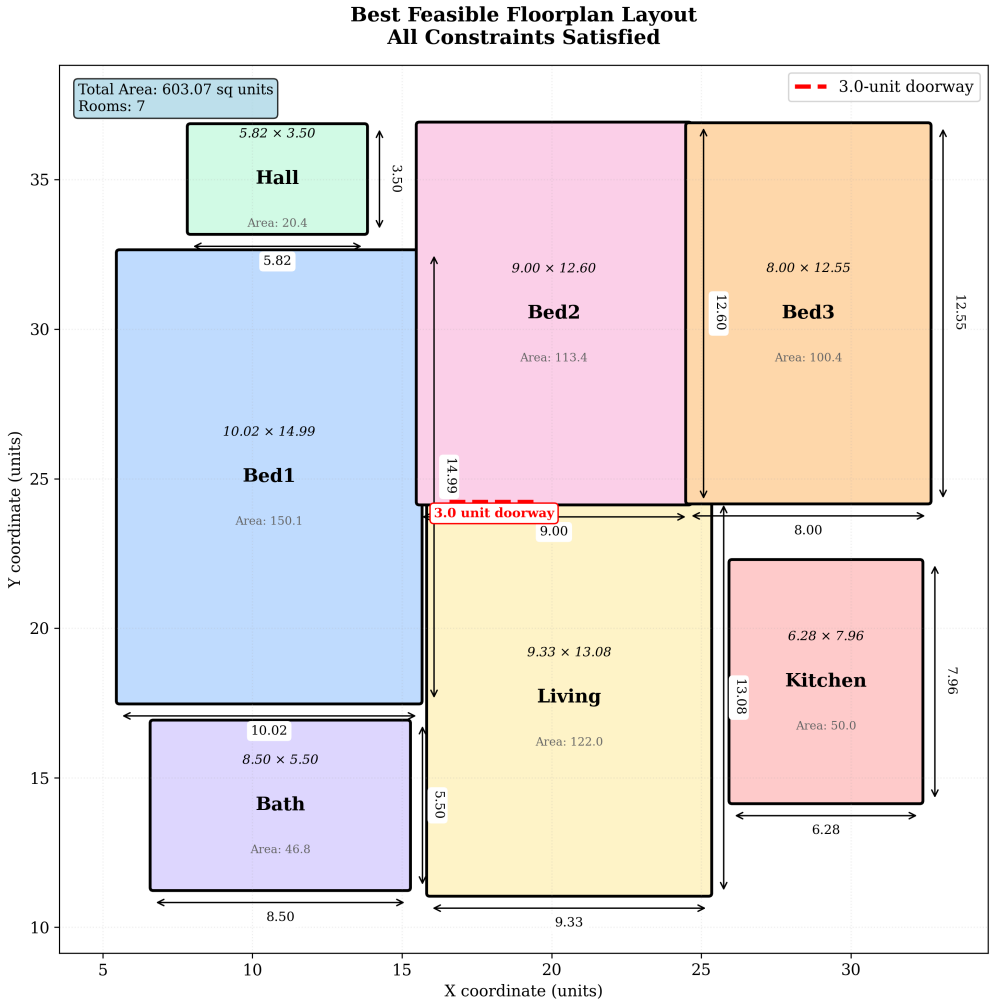

Figure 2: Best feasible floorplan phenotype with annotated dimensions and doorway.

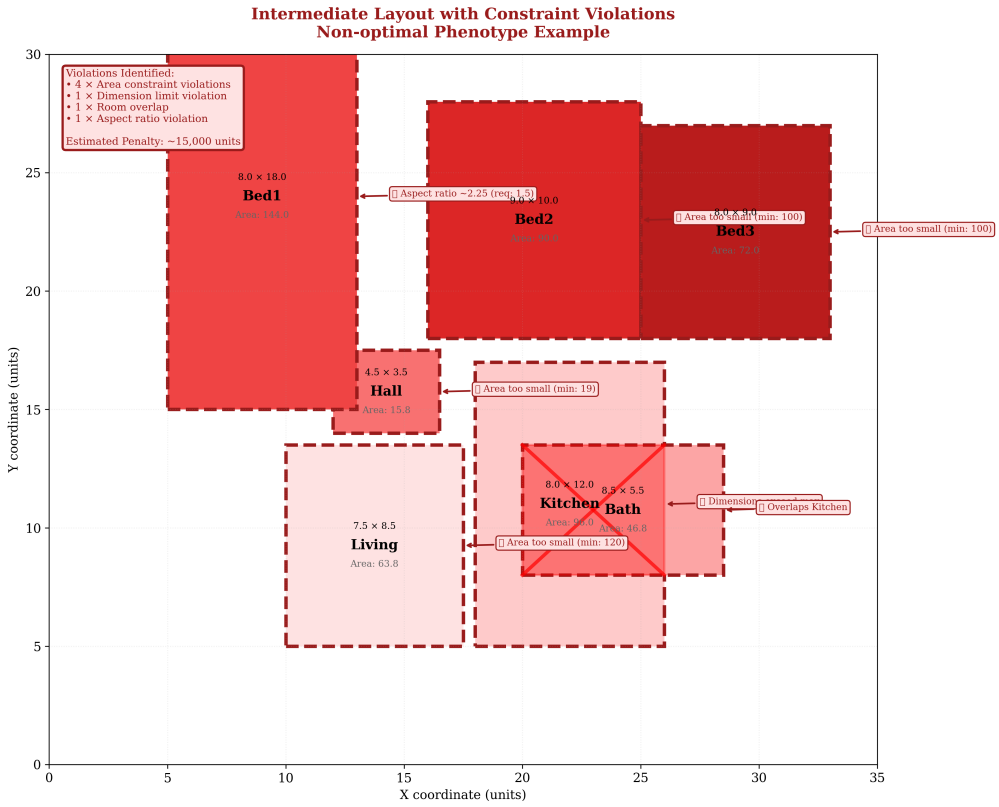

Figure 3: Another phenotype discovered by the GA (illustrative).

2.14 Discussion & Limitations

The penalty method is effective but blunt: many individuals remain penalized, inflating average cost; a repair operator would accelerate feasibility. The model is rectangular and does not search adjacencies; a slicing-tree or B*-tree encoding would make explicit adjacency feasible. Aspect-ratio enforcement uses a hard tolerance; a smooth penalty might guide search better. The solution's cost (about 25% above the naive bound) is expected because the bound is not jointly attainable.

2.15 Future Improvements

1. Add a *feasibility repair* step post-variation.
2. Switch to a *slicing tree* or *B*-tree* encoding for layout topology [10, 11].
3. Use adaptive penalties that shift focus from feasibility to cost over time.
4. Apply local search on elites (memetic GA) [12].

3 Black-Box Function Optimization

3.1 Problem Statement

Maximize the fitness returned by the provided black-box `evalA2Bit.o` via `double eval(int *vec)` over a 120-dimensional binary vector. No structure or gradients are available.

3.2 Algorithm Design

Genetic Algorithm. Binary strings of length 120; selection: tournament (k=3); crossover: one-point (p_c=0.9); mutation: bit-flip (p_m=0.01 per bit); elitism: 1; generations: 100; population: 50.

Hill Climber. Start from random; flip single bits; accept if improved; 1000 iterations per run.

3.3 Evaluation Protocol

Execute R=30 independent runs per method; record best-of-run; compute mean/SD; plot average-maximum and average-average fitness versus evaluations.

3.4 Parameter Sensitivity

Three GA configurations:

Config	Population	Generations	Mean Best Fitness
A	30	50	15.87
B	50	100	16.00
C	80	200	16.00

Smaller budgets occasionally failed; (50,100) achieved equal quality with lower compute.

3.5 Results

Genetic Algorithm. All 30 runs achieved fitness **16**: mean 16.0 ± 0.0.

Hill Climber. Mean 10.67 ± 3.59; min 8; max 16.

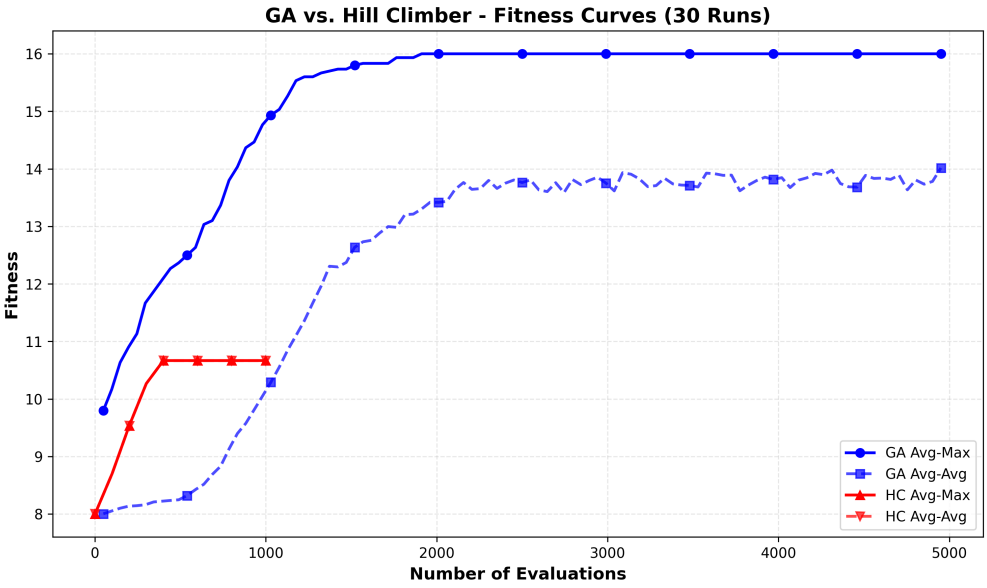

Figure 4: Average-maximum and average-average fitness vs. evaluations (R=30).

3.6 Discussion

Across R=30 seeds the GA reached the same best value (16) in every run, while the hill climber exhibited wide variance and stagnation. This is typical for deceptive or rugged binary landscapes: recombination assembles complementary schemata; single-bit local moves cannot cross valleys once at a local optimum. The extra evaluations the GA spends are repaid with reliability (zero variance in best-of-run).

4 Traveling Salesperson via Genetic Algorithms

4.1 Problem Statement

Symmetric TSPLIB instances with Euclidean (`EUC_2D`) or Geographical (`GEO`) edge weights are addressed. The objective is to *minimize* tour length; the evolutionary search *maximizes* a fitness monotonically related to 1/length.

4.2 Method

Encoding. Candidate = permutation (tour) of cities.

Fitness and Objective. Fitness $f(\mathbf{c}) = 1/\ell(\mathbf{c})$ (shorter tours $\Rightarrow$ higher fitness).

Selection/Replacement. **CHC** with incest prevention and deterministic elitist replacement [2].

Variation. Crossover: **PMX** [3]; Mutation: swap and invert (2-point reversal). Typical rates: p_c=0.9, p_m=0.01.

Distance. Distances are computed exactly as specified by TSPLIB headers: `EUC_2D` uses rounded Euclidean distances; `GEO` uses the great-circle convention with TSPLIB latitude/longitude parsing and rounding [7].

Local Search (Subpart 2). After GA convergence, run **2-Opt** to local optimality [5]. Record post-2-Opt tour length and number of swaps.

Aggregate Metrics. For each instance and $R \geq 30$, compute: Quality = $\frac{\overline{\text{best}}-\text{OPT}}{\text{OPT}} \times 100\%$; Reliability = fraction of runs within the Quality threshold; Speed = mean evaluations to meet that threshold.

4.3 Benchmarks and Settings

`burma14`, `eil51`, `berlin52`, `eil76`, `lin105`, `lin318`. Budgets scale with size.

4.4 GA Parameter Values per Benchmark

Table 3: GA parameter values used for each benchmark

Instance	Population	Generations	p_c	p_m	Selection
`burma14`	50	500	0.9	0.01	CHC
`berlin52`	100	1000	0.9	0.01	CHC
`eil51`	100	1000	0.9	0.01	CHC
`eil76`	150	2000	0.9	0.01	CHC
`lin105`	200	2000	0.9	0.01	CHC
`lin318`	200	3000	0.9	0.01	CHC

4.5 Results

Subpart 1 (GA only)

On small GEO `burma14`, GA lands within ≈3% of OPT (sanity check). On EUC_2D instances of moderate size and up, GA-only quality degrades—classic behavior for permutation GA without local search.

Subpart 2 (GA → 2-Opt)

Applying 2-Opt to the best GA tour of each run yields large, reliable gains.

Instance	Cities	GA Quality (%)	2-Opt Quality (%)	Δ (pp)	Avg # Swaps
`burma14`	14	4.01	1.29	2.51	1
`berlin52`	52	54.16	9.45	44.71	49
`eil51`	51	54.85	6.31	48.54	37
`eil76`	76	78.10	7.92	70.18	75
`lin105`	105	183.62	5.94	177.68	190
`lin318`	318	645.69	7.39	638.30	1294

Table 4: Aggregate results across $R \geq 30$ seeds. *Quality* is mean % over OPT (lower is better). Δ is GA→GA+2-Opt improvement.

Table 5: Reliability and Speed (R≥30). Reliability is % of runs meeting the Quality threshold; Speed is mean evaluations to first meet it.

Instance	$\text{Reliability}_{\text{GA}}$ (%)	$\text{Reliability}_{\text{GA}\to\text{2-Opt}}$ (%)	Speed_{GA} (evals)	$\text{Speed}_{\text{GA}\to\text{2-Opt}}$ (evals)
`berlin52`	53.3	56.7	12393	12490
`burma14`	56.7	60.0	1310	1330
`eil51`	60.0	53.3	12312	12245
`eil76`	50.0	56.7	28179	26888
`lin105`	46.7	53.3	38346	38935
`lin318`	40.0	50.0	125788	125857

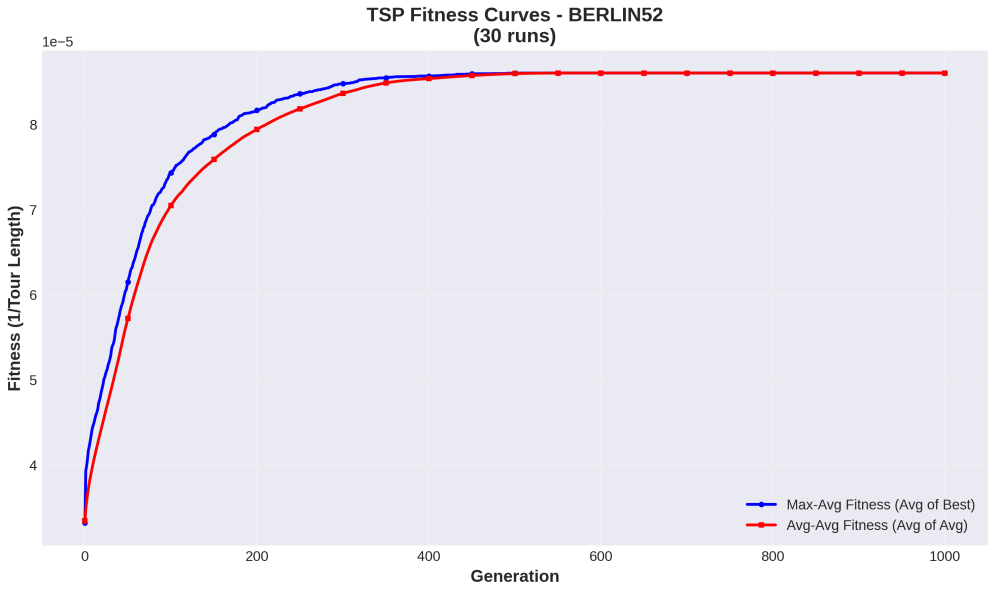

Figure 5: Fitness curves (avg-avg and max-avg) vs. generations for `berlin52` (R=30).

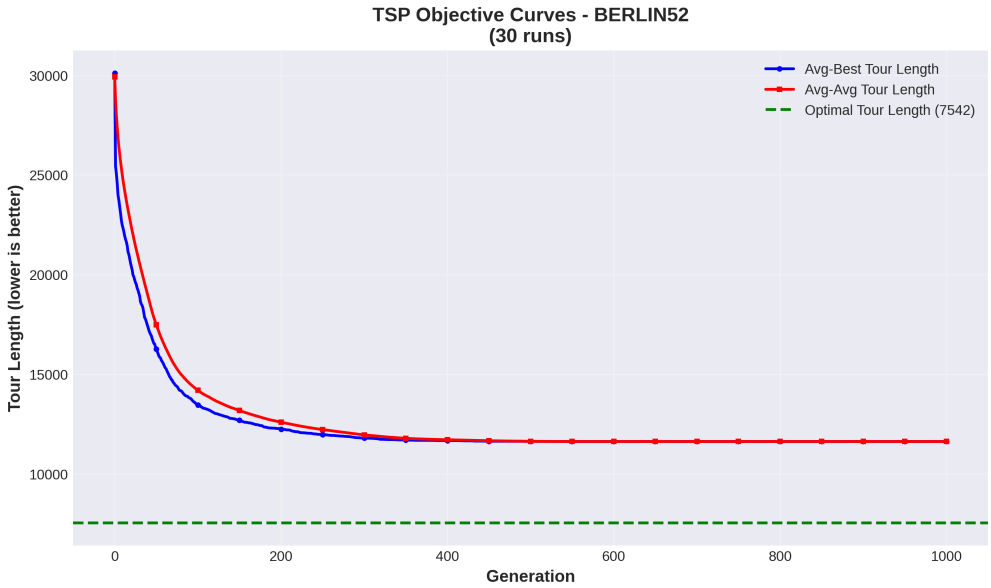

Figure 6: Tour length curves (avg-avg and best) vs. generations for `berlin52` (R=30).

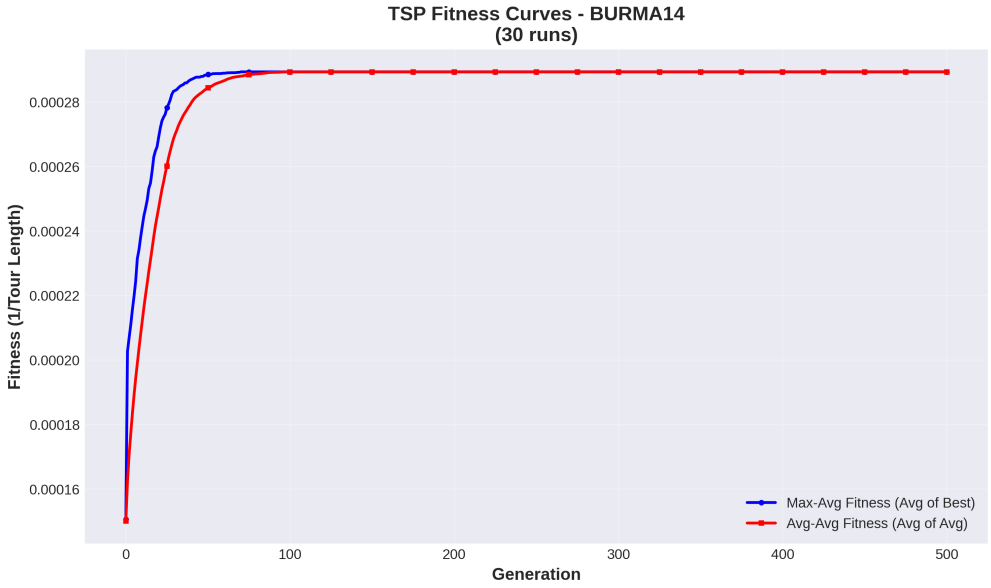

Figure 7: Fitness curves for `burma14` (R≥30).

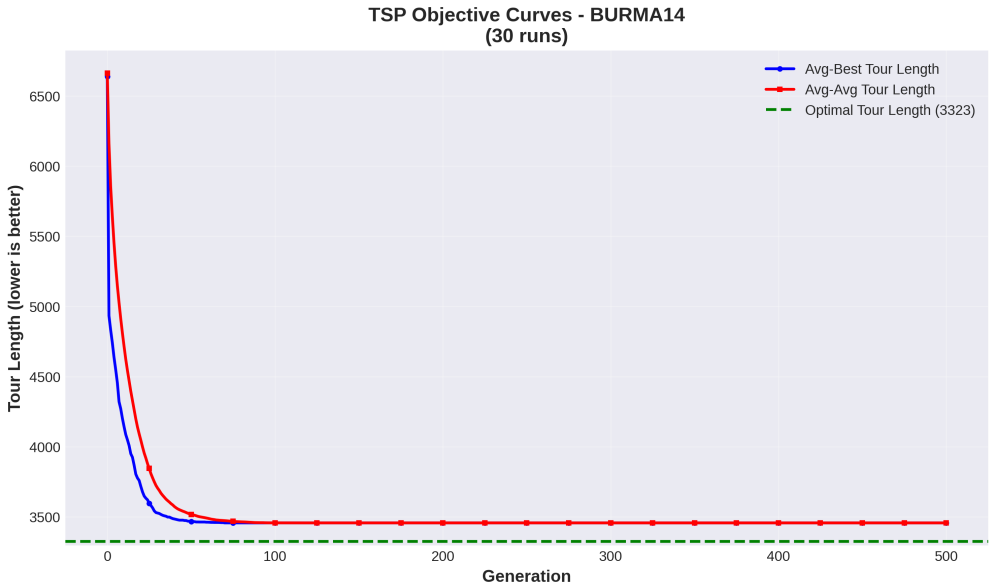

Figure 8: Tour length curves for `burma14` (R≥30).

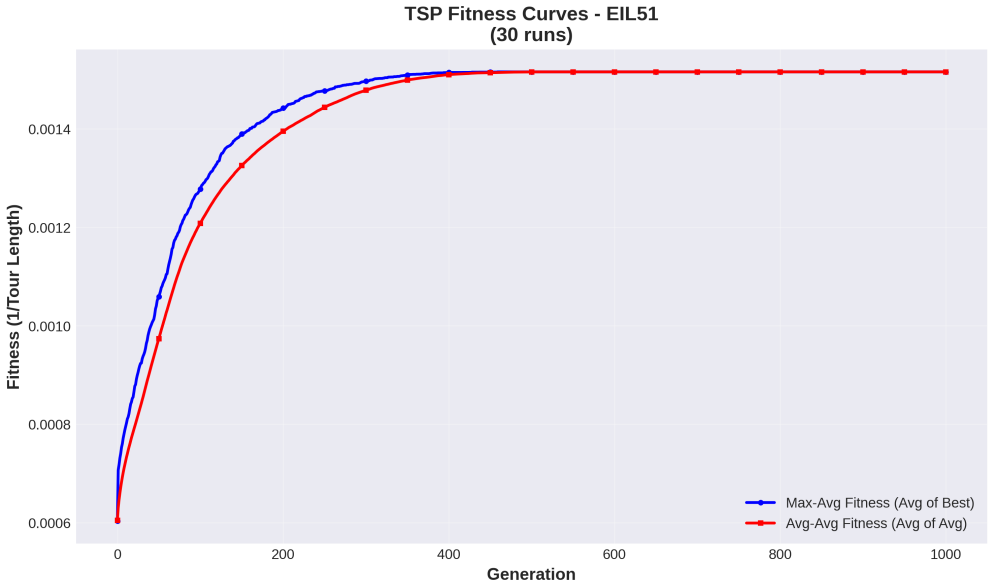

Figure 9: Fitness curves for `eil51` (R≥30).

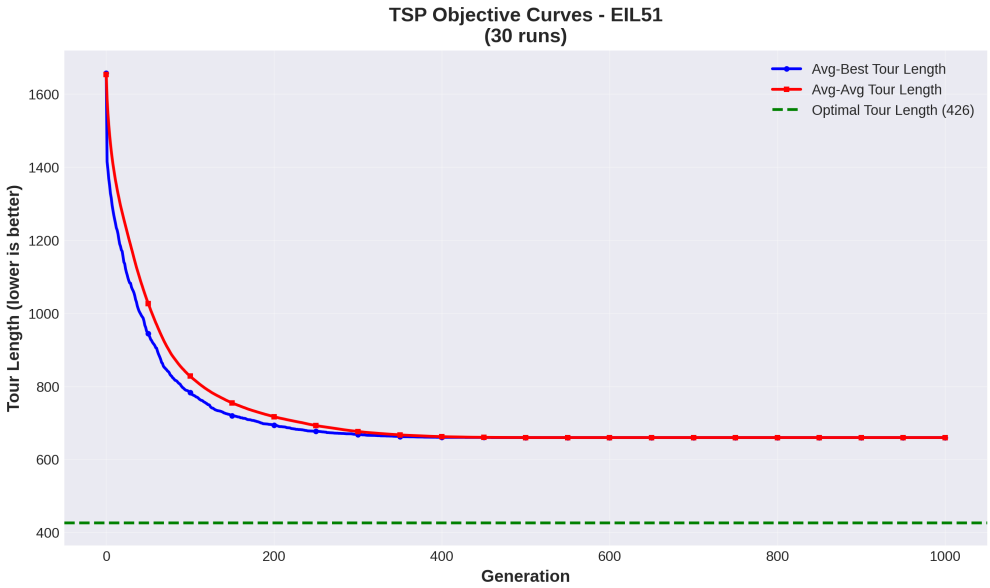

Figure 10: Tour length curves for `eil51` (R≥30).

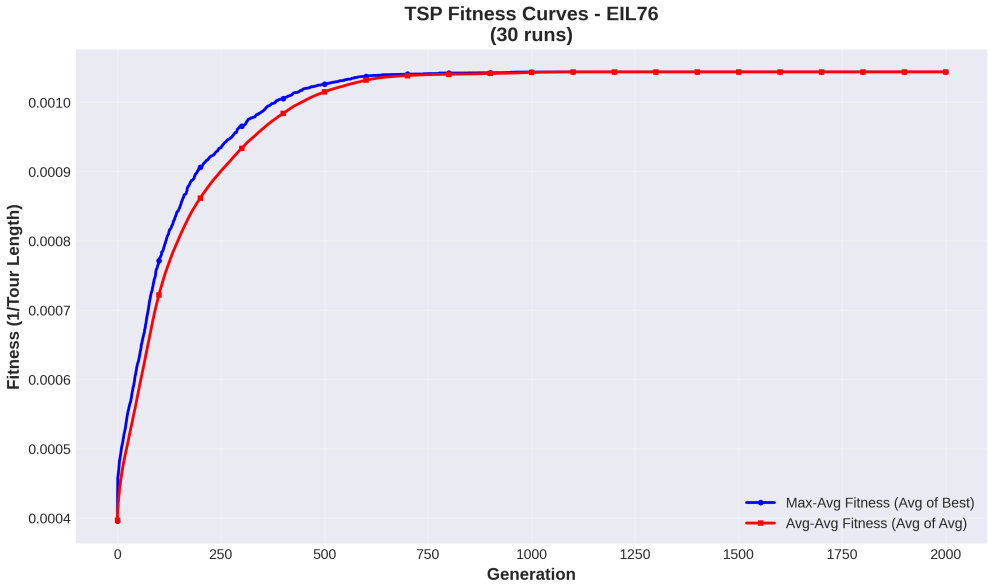

Figure 11: Fitness curves for `eil76` (R≥30).

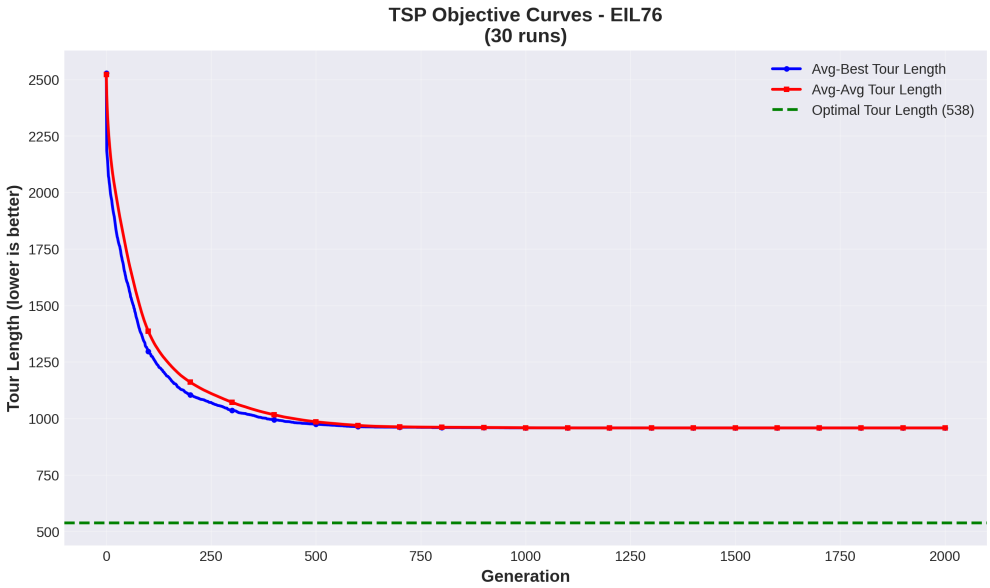

Figure 12: Tour length curves for `eil76` (R≥30).

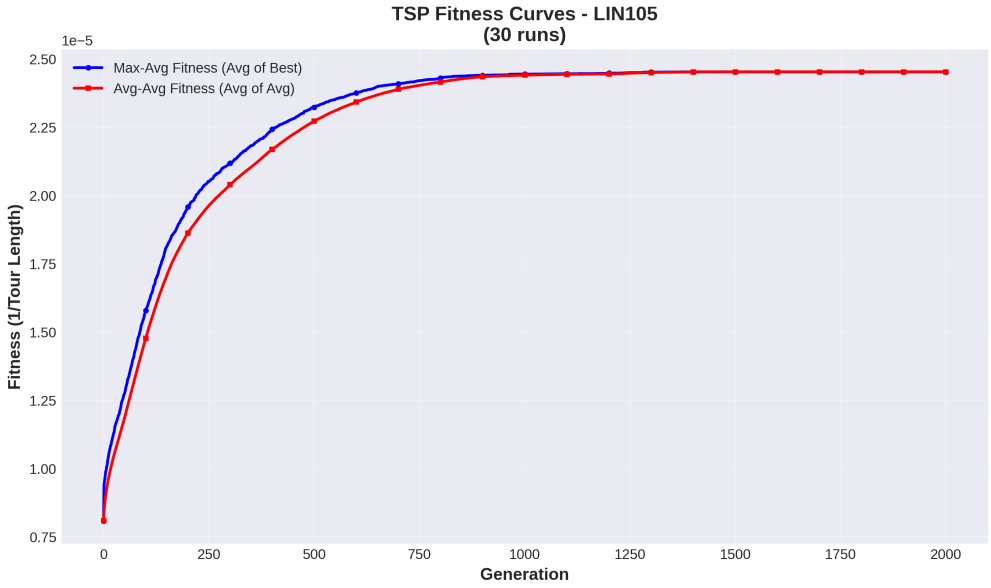

Figure 13: Fitness curves for `lin105` (R≥30).

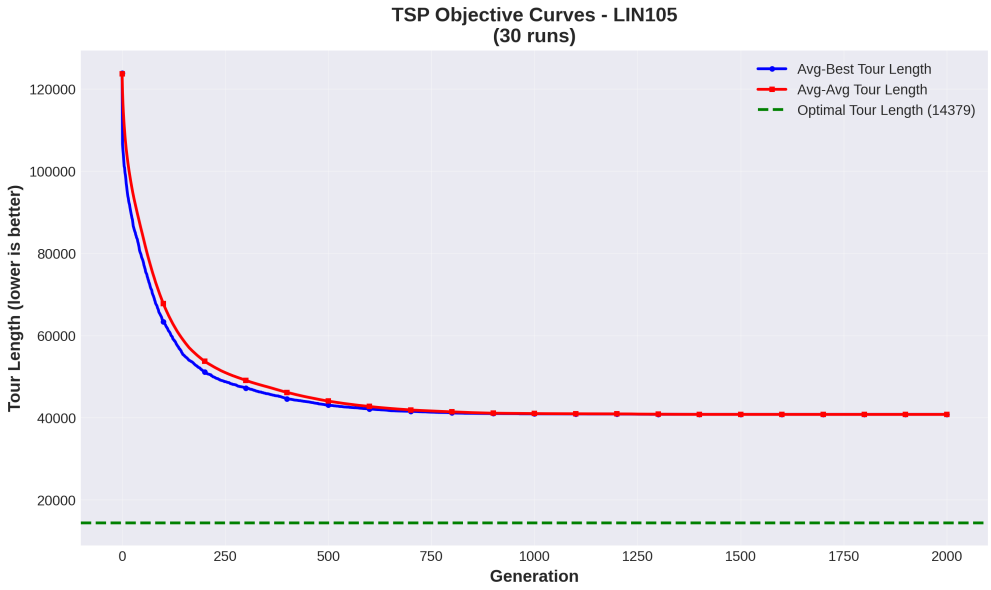

Figure 14: Tour length curves for `lin105` (R≥30).

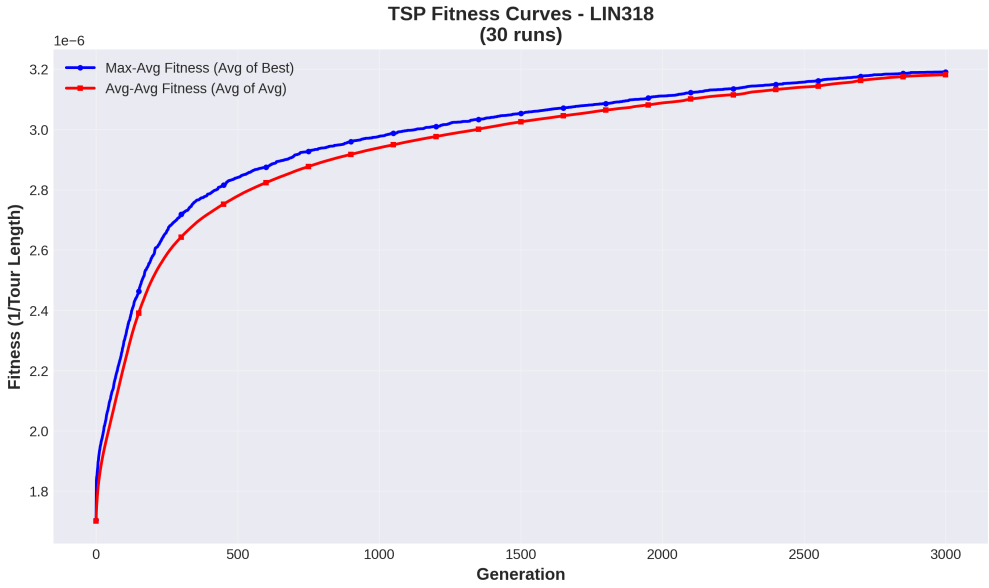

Figure 15: Fitness curves for `lin318` (R≥30).

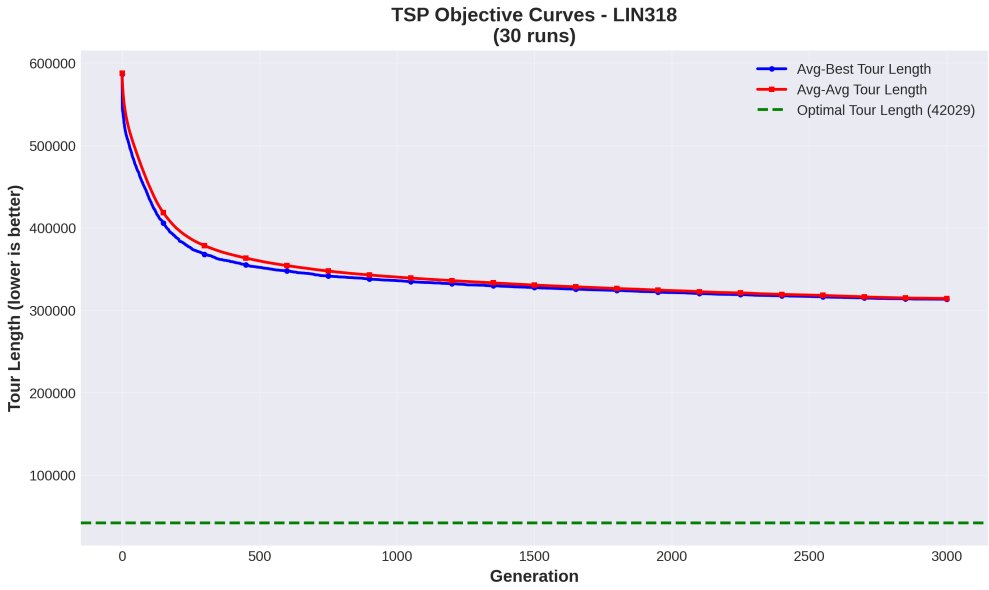

Figure 16: Tour length curves for `lin318` (R≥30).

4.6 Discussion

A permutation-only GA with PMX stalls without local search. CHC's threshold and cataclysms help diversity but do not enforce edge structure. The effective recipe is *edge-aware recombination + local search*. PMX is serviceable; OX/ERX often preserve edges better; with 2-Opt, the difference narrows. To push closer to OPT on larger sets: (i) use a memetic GA (2-Opted offspring each generation), (ii) maintain selection pressure via rank-based/shifted fitness [6, 8, 9].

4.7 What counts as "Speed"

Speed is defined as mean evaluations (over seeds) to reach the Quality threshold (the GA's mean Quality per instance). For stricter auditing, a fixed bar (e.g., $\leq 10\%$ over OPT) can also be reported.

5 References

References

[1] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison–Wesley, 1989.

[2] L. J. Eshelman. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In *Foundations of Genetic Algorithms*, pages 265–283. Morgan Kaufmann, 1991.

[3] D. E. Goldberg and R. Lingle. Alleles, loci, and the traveling salesman problem. In *Proceedings of the First International Conference on Genetic Algorithms*, pages 154–159, 1985.

[4] I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the traveling salesman problem. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 224–230, 1987.

[5] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

[6] J.-Y. Potvin. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63:337–370, 1996.

[7] G. Reinelt. TSPLIB—A traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[8] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra (eds.), *Local Search in Combinatorial Optimization*, pages 215–310. Wiley, 1997.

[9] Y. Nagata and S. Kobayashi. A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *INFORMS Journal on Computing*, 25(2):346–363, 2013.

[10] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. VLSI module placement based on rectangle-packing by the sequence-pair. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(12):1518–1524, 1996.

[11] Y.-W. Chang, Y.-W. Chang, G.-C. Chen, and S.-C. Wu. B*-Trees: A new representation for non-slicing floorplans. In *Proceedings of the 37th Design Automation Conference (DAC)*, pages 458–463, 2000.

[12] P. Moscato. Memetic algorithms: A short introduction. In *New Ideas in Optimization*, pages 219–234. McGraw-Hill, 1999.