GEN5_V5 (CVRP3) HANDOFF NOTE

This document is a ?pick up from here? snapshot for the Gen5_v5 solver + runner + viewer that we iterated on in this thread.

1) WHAT TO RUN (MAIN ENTRYPOINTS)

CORE SOLVER PIPELINE

- ‘gen5_multi_axis_solver_schedule_validated.py‘ ? the *runner/scheduler* (CLI you invoke). Loads a ‘.vrp‘, optionally a BKS ‘.sol‘, runs staged solve blocks, logs to SQLite, and writes the final ‘.sol‘ output.
- ‘gen5_multi_axis_solver_validated.py‘ ? the *solver module* (loaded via ‘--solver-path‘). Contains STN building, multi-axis neighbor logic, relocate/2-opt kernels, and EPN top-up implementation.

SOLVER STRATEGIES / FEATURES

- ‘gen5_multi_axis_solver_schedule_validated.py‘ ? stage loop + axis schedule + plateau hooks + strict-feasible snapshot tracking.
- ‘gen5_multi_axis_solver_validated.py‘ ? EPN (edge proximity neighbors), geo KNN1, STN ranking knobs, and the local search primitives.

VIEWER

- ‘viewer/backend.py‘ ? FastAPI server that reads ‘runs.sqlite‘ and serves:
- run list page
- run detail page with stage table + plots + route selectors
- ‘viewer/gen5_v5_runs.html‘ ? run list UI (filters, pick run_id).
- ‘viewer/gen5_v5_run.html‘ ? run detail UI (stage progress table, solver/BKS plots, route toggles, delta view).

2) LOGGING (ENABLED BY DEFAULT)

- Default DB: ‘runs.sqlite‘ (override with ‘--db‘).
- Every solver invocation gets a ‘run_id‘ in SQLite.
- CLI supports ‘--notes‘ to tag a run (stored in DB; shown in viewer).
- Output ‘.sol‘ is auto-prefixed with the run id: ‘out/runid_<run_id>_<your_out_sol_name>.sol‘

DB tables used by Gen5_v5:
- ‘gen5_v5_runs‘ ? one row per run (instance, args, notes, timestamps).
- ‘gen5_v5_snapshots‘ ? stored plans/costs per stage boundary (start/end).
- ‘gen5_v5_results‘ ? final summary row for the run (best strict cost, wall time, etc).

3) STAGES AND ?BEST? TRACKING (IMPORTANT)

The runner tracks two notions of ?best?:
- ‘best_strict‘: best solution feasible under original capacity ‘Q_orig‘ (this is what the run will output).
- ‘best_any‘: best solution under the *current stage?s effective capacity* (e.g., soft stage ‘Q_eff = soft_factor * Q_orig‘). This may violate ‘Q_orig‘.

Console progress lines:
- ‘best=...‘ is the **best strict-feasible cost so far**.
- ‘best_any=...‘ may appear during soft stages when a softer-capacity solution beats ‘best_strict‘.

Final output ‘.sol‘:
- Always written from ‘best_strict‘ (prevents ?time limit ends mid-block ? output regresses?

bugs).

4) MAJOR CLI PARAMETERS (RUNNER)

REQUIRED / COMMON

- positional ‘vrp‘: instance path.
- ‘--out-sol out/<name>.sol‘: output path (will be prefixed with ‘runid_XX_‘).
- ‘--solver-path gen5_multi_axis_solver_validated.py‘
- ‘--db runs.sqlite‘
- ‘--notes "..."‘ (recommended)

BUDGETS / PROGRESS

- ‘--time-limit-s <seconds>‘: wall-clock cap (0 = no limit).
- ‘--progress-every-s <seconds>‘: progress print cadence.

LOOKAHEAD (BEAM WIDTH)

- ‘--strict-lookahead <int>‘: strict/tight stages (default 1).
- ‘--soft-lookahead <int>‘: soft stage (default 5).

AXIS / PHI SCHEDULE

- ‘--phis "45,-45,90,0,135,-135"‘: candidate-axis angles in degrees.
- ‘--axis-count-schedule "1,2,3,4,5,6,5,4,3,2,1"‘: active axis count per block (prefix of ‘--phis‘).
- ‘--axis-block-iters <int>‘: iterations per axis-count block.
- ‘--axis-block-time-s <float>‘: seconds per block (0 disables; uses iters).
- ‘--axis-schedule-mode cycle|prune‘: how to rotate phis inside a block.
- ‘--axis-block-seed-step <int>‘: deterministic seed increment per block.

STAGE SCHEDULE (TIGHT/SOFT/STRICT)

- ‘--tight-factor <float>‘: capacity multiplier for tight stage (default 0.7).
- ‘--soft-factor <float>‘: capacity multiplier for soft stage (default 1.8).
- ‘--tight-iters‘, ‘--soft-iters‘, ‘--strict-iters‘: block iteration budgets.
- ‘--use-schedule‘: enable tight?soft?strict loop schedule.

EPN (ENABLED BY DEFAULT IN SOFT)

- ‘--no-epn-soft‘: disable EPN top-up in soft stage.
- ‘--epn-k1‘: KNN1 size for exclusion.
- ‘--epn-k‘: max nodes returned per edge query.
- ‘--epn-max-tested‘: cap tested nodes per edge query.
- ‘--epn-long-edges-per-route‘: trigger edges per route (long edges).
- ‘--epn-max-moves‘: applied EPN moves per LNS step.
- ‘--epn-d2-beta‘: edge-near cutoff: ‘d2(w, segment) <= beta * edge_len2‘.

FULL PER-ROUTE 2-OPT ?UNCROSSING? CLEANUP

- ‘--full-2opt-on-best‘: run a full per-route 2-opt local search when a new global best is found.
- ‘--full-2opt-stage strict|all‘: apply only when stage is strict, or in all stages.
- ‘--full-2opt-max-len <int>‘: apply only to routes with length ? this (0 = no cap).
- ‘--full-2opt-max-passes <int>‘: passes per route.

STN ?FAR NODE? KNOBS (EUC_2D ONLY)

These are only active for coordinate-based EUC_2D instances; for EXPLICIT instances the runner will warn and ignore them.
- ‘--stn-r2-penalty-frac‘: STN ranking penalty for mismatched depot-radius (‘r2‘) (pushes

?similar radial band? neighbors).
- ‘--stn-far-y-mult‘, ‘--stn-far-r-mult‘: increase y/r pool sizes for far nodes to widen candidate neighborhoods.
- ‘--stn-profile-schedule "far,lateral"‘ + ‘--stn-profile-blocks‘: cycle different STN profiles across axis blocks.
- ‘--stn-lateral-*‘: profile-specific knobs for ?lateral? variant.

OTHER KNOBS YOU MAY SEE IN OLD COMMANDS

- ‘--k-remove‘: remove-list size used by a destroy-like LNS step; optional and can be omitted for baseline strict runs.
- ‘--ffs-on-plateau‘ + related ‘--ffs-*‘: farthest-first seeding plateau breaker (optional; can degrade if overused).

5) COMMANDS YOU CAN REUSE (EXAMPLES)

A-N32-K5 (SINGLE PHI STRICT-ONLY BASELINE)

‘‘‘bash
python3 gen5_multi_axis_solver_schedule_validated.py \
/Users/bharatmudholkar/projects/codex_projects/cvrp3/cvrp_data_instances/A/A-n32-k5.vrp \
--bks /Users/bharatmudholkar/projects/codex_projects/cvrp3/cvrp_data_instances/A/A-n32-k5.sol \
--out-sol out/A-n32-k5.phi45.strict.sol \
--solver-path gen5_multi_axis_solver_validated.py \
--phis "45" \
--time-limit-s 300 \
--strict-iters 100000000 \
--steps-reloc 200 --steps-2opt 160 \
--strict-lookahead 3 \
--progress-every-s 5 \
--db runs.sqlite \
--notes "A-n32-k5 phi=45 strict-only lookahead=3 t=300s"
‘‘‘

LOGGI-N1001-K31 (MULTI-PHI + AXIS-COUNT SCHEDULE, BASELINE STRICT)

‘‘‘bash
python3 gen5_multi_axis_solver_schedule_validated.py \
/Users/bharatmudholkar/projects/codex_projects/cvrp3/cvrp_data_instances/DIMACS/Loggi-n1001-k31.vrp
  \
--bks
  /Users/bharatmudholkar/projects/codex_projects/cvrp3/cvrp_data_instances/DIMACS/Loggi-n1001-k31.sol
  \
--out-sol out/Loggi-n1001-k31.axes_1to6to1.strict.sol \
--solver-path gen5_multi_axis_solver_validated.py \
--phis "45,-45,90,0,135,-135" \
--axis-count-schedule "1,2,3,4,5,6,5,4,3,2,1" \
--axis-block-iters 20 \
--axis-schedule-mode cycle \
--axis-block-seed-step 1000 \
--time-limit-s 300 \
--strict-iters 100000000 \
--steps-reloc 60 --steps-2opt 40 \
--strict-lookahead 3 \
--progress-every-s 5 \
--db runs.sqlite \
--notes "Loggi-n1001-k31 strict + axis schedule 1-6-1; t=300s"
‘‘‘

If NumPy is missing in your venv, install it before running:
‘‘‘bash
python3 -m pip install numpy

‘‘‘

6) VIEWER USAGE (DO NOT REBUILD A NEW VIEWER)

Start the backend:
‘‘‘bash
python3 viewer/backend.py
‘‘‘

Open in browser:
- Run list: ‘http://localhost:8000/gen5v5‘
- Run detail: click a run, or ‘http://localhost:8000/gen5v5/run?run_id=<id>‘

Detail page layout:
1) Stage progress table (per snapshot): includes cost, gap vs BKS (if provided), edge-match metrics (if available), and effective ‘Q‘.
2) Two plots: Solver vs BKS (only these two; designed to handle large N by drawing a polyline per route).
3) Route list with:
- select/deselect all
- per-route toggles
- delta mode (show only changed routes vs previous snapshot)
- ?show violated only? + highlight overshoot amount when ‘load > Q‘.

7) KNOWN GOTCHAS (AS OF THIS HANDOFF)

- **STN profile scheduling is EUC_2D-only**: if the instance is EXPLICIT, the runner warns and ignores ‘--stn-profile-schedule‘ and far/lateral knobs.
- **Gap in final summary vs progress**: progress prints ‘best_strict‘; the final ‘.sol‘ should be written from ‘best_strict‘. If you still see a mismatch, check that the final output path is the ‘runid_XX_...‘ file (not an older file), and confirm the run is using the latest runner.
- **BKS mismatches**: if ‘--bks‘ has node IDs not present in the instance (e.g., wrong file), BKS comparison will be disabled with a warning rather than crashing.

8) WHERE TO CONTINUE NEXT TIME

Recommended next investigation items:
- Make the ?phi switching? more productive (ensure each phi gets enough local optimization before switching).
- If Loggi instances are EXPLICIT, extend candidate/STN profile alternation logic to EXPLICIT mode (or provide a coordinate proxy).
- Add a route-level stronger uncrossing/cleanup stage if crossings remain after full 2-opt (typically crossings left are across routes).